

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA

19960430 056



THESIS

COMPUTER SIMULATION OF AN UNMANNED AERIAL VEHICLE ELECTRIC PROPULSION SYSTEM

by

Joel Yourkowski

March 1996

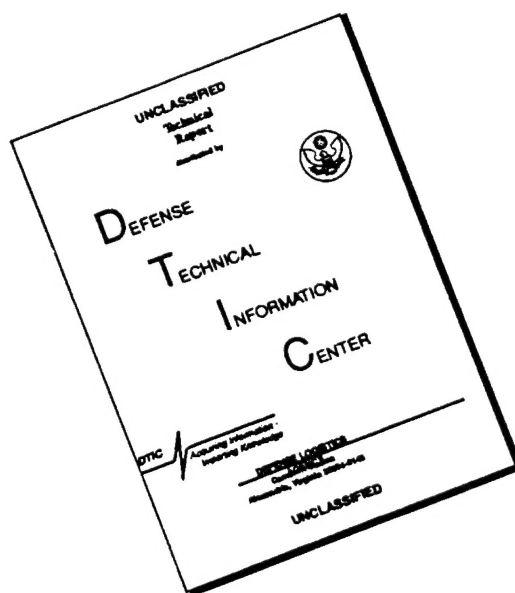
Thesis Advisor:

Jovan E. Lebaric

Approved for public release; distribution is unlimited

DTIC QUALITY INSPECTED 1

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE COMPUTER SIMULATION OF AN UNMANNED AERIAL VEHICLE ELECTRIC PROPULSION SYSTEM		5. FUNDING NUMBERS	
6. AUTHOR(S) Yourkowski, Joel			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Research Laboratory Washington, D.C. 20375		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) There has been a substantial increase in the use of electric propulsion systems in Unmanned Aerial Vehicles (UAVs). However, this area of engineering has lacked the benefits of a dynamic model that could be used to optimize the design, configurations and flight profiles. The Naval Research Laboratory (NRL) has accurate models for the aerodynamics associated with UAVs. Therefore the proposed electric propulsion model would use the torque and RPM requirements generated by the aerodynamic model and provide an accurate representation of the desired UAV electric propulsion system. This thesis reports on the development of such a model. The model is adaptive in the sense that motor and battery parameters can be altered by the user to reflect systems currently in use or those considered for future systems. Not only will the simulation model accurately reflect the operating conditions of the motor and battery during the mission, but different flight profiles with the same configuration can be evaluated in terms of efficiency based on the Percent Battery Capacity Used (PBCU) at the end of the mission. This Electric Propulsion Simulator is part of a larger NRL project intended to design and deliver UAVs to the Naval Service over the next few years.			
14. SUBJECT TERMS UAV, Electric Vehicles, Electric Machine, Simulation, Electric Propulsion		15. NUMBER OF PAGES 122	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Approved for public release; distribution is unlimited.

**COMPUTER SIMULATION OF AN UNMANNED AERIAL VEHICLE
ELECTRIC PROPULSION SYSTEM**

Joel Yourkowski
Major, United States Marine Corps
B.S., United States Naval Academy, 1983

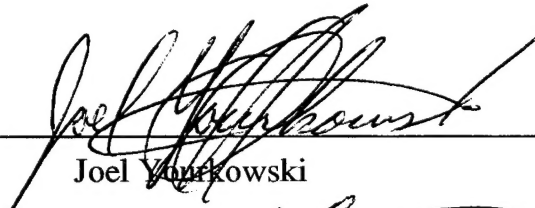
Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

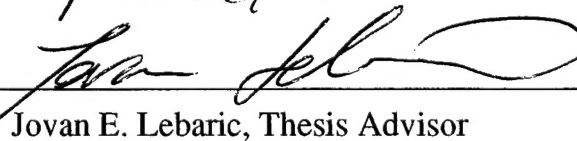
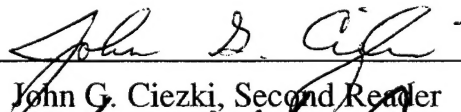
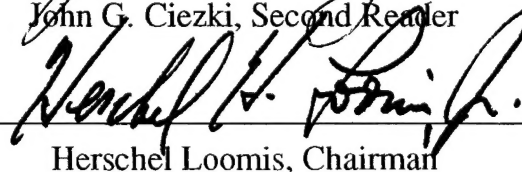
from the

**NAVAL POSTGRADUATE SCHOOL
March, 1996**

Author:


Joel Yourkowski

Approved by:


Jovan E. Lebaric, Thesis Advisor
John G. Ciezki, Second Reader
Herschel Loomis, Chairman

Department of Electrical and Computer Engineering

ABSTRACT

There has been a substantial increase in the use of electric propulsion systems in Unmanned Aerial Vehicles (UAVs). However, this area of engineering has lacked the benefits of a dynamic model that could be used to optimize the design, configurations and flight profiles. The Naval Research Laboratory (NRL) has accurate models for the aerodynamics associated with UAVs. Therefore the proposed electric propulsion model would use the torque and RPM requirements generated by the aerodynamic model and provide an accurate representation of the desired UAV electric propulsion system. This thesis reports on the development of such a model. The model is adaptive in the sense that motor and battery parameters can be altered by the user to reflect systems currently in use or those considered for future systems. Not only will the simulation model accurately reflect the operating conditions of the motor and battery during the mission, but different flight profiles with the same configuration can be evaluated in terms of efficiency based on the Percent Battery Capacity Used (PBCU) at the end of the mission. This Electric Propulsion Simulator is part of a larger NRL project intended to design and deliver UAVs to the Naval Service over the next few years.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. BACKGROUND	1
B. THESIS OVERVIEW	1
1. Modeling the Electrical Components	2
2. Simulation Software and Model Development	2
3. Model Testing and Evaluation	3
4. Conclusions and Future Work	3
II. MODELING THE ELECTRICAL COMPONENTS	5
A. MODEL OVERVIEW	5
B. THE PERMANENT-MAGNET BRUSHLESS DIRECT CURRENT (PMBDC) MACHINE	6
1. The Development of Motor State Variables	6
2. Voltage, Current and Power Equations	11
3. Accounting for Copper Losses in the Armature Windings	12
C. THE BATTERY COMPONENTS	14
1. Percent Battery Capacity Used (PBCU) vs Current	15
2. Battery Voltage vs PBCU and Current	16
III. SIMULATION SOFTWARE AND MODEL DEVELOPMENT	17
A. SIMULATION SOFTWARE	17
B. MOTOR CONTROL BLOCK	19
1. PMBDC Motor Block	20
2. Equivalent Torque Block	20
3. Torque Adjustment Block	21
4. Motor Efficiency Block	23
C. BATTERY DISCHARGE BLOCK	24
D. VOLTAGE CALCULATIONS BLOCK	25
E. DATA COLLECTION AND PLOTTING BLOCK	26
F. SIMPLIFIED KRAUSE MODEL	27
IV. MODEL TESTING AND EVALUATION	29
A. OVERVIEW OF THE TESTING APPROACH	29
1. Specification of the Initial Simulation Parameters	29
B. TEST RESULTS	32
1. Torque and RPM Accuracy	32
2. Current and Voltage Accuracy	35
3. Efficiency Comparisons	35

C. SIMULATOR OUTPUT	38
1. Simulation Plots	38
2. MATLAB Command Window Information	38
V. CONCLUSIONS	41
A. CURRENT APPLICATION	41
B. FUTURE WORK	42
1. Overall System	42
2. Motor Model	42
3. Battery Model	42
4. Future Applications	43
APPENDIX A. SIMULATOR INSTRUCTIONS	45
A. STARTING THE ELECTRIC PROPULSION SIMULATOR	45
B. DESCRIPTIONS OF THE EPS CONTROL BUTTONS	45
C. STEP SEQUENCE IN OPERATING THE EPS	50
D. CRITICAL VARIABLES DESCRIPTION	51
APPENDIX B. MATLAB PROGRAMS	55
APPENDIX C. MANUFACTURER'S DATA SHEETS	99
LIST OF REFERENCES	109
INITIAL DISTRIBUTION LIST	111

ACKNOWLEDGMENT

The author would like to thank Professon Jovan Lebaric for his help and enthusiasm on this project.

A sincere thanks also to Richard Foch and Barry Walden of Naval Research Laboritory for sponsoring the research that went into the Electric Propulsion Simulator.

This project would have never been completed without the support and sacrifice of my lovely wife and my three delightful daughters. My love and grattitude toward them is beyond what words can describe.

Finally, I give praise and honor to the Lord Jesus Christ who has been and is and will be my rock, my shield and my Savior. For apart from Him I can do nothing.

I. INTRODUCTION

A. BACKGROUND

There has been a substantial increase in the use of electric propulsion systems in Unmanned Aerial Vehicles (UAV). However, this area of engineering has lacked the benefits of a dynamic model that could be used to optimize design, configurations and flight profiles. The Naval Research Laboratory (NRL) has accurate models for the aerodynamics associated with UAVs. Therefore the proposed electric propulsion model would use the torque and RPM requirements generated by the aerodynamic model and provide an accurate representation of the desired UAV electric propulsion system. The model should be adaptive in the sense that motor and battery parameters could be altered by the user to reflect systems currently in use or possible designs of future systems. Not only would the simulation model accurately reflect the operating conditions of the motor and battery during the mission, but different flight profiles with the same configuration could be evaluated in terms of efficiency based on the Percent Battery Capacity Used (PBCU) at the end of the mission. Since the propulsion model responds to the requirements of the aerodynamic model there is no way to adjust the torque and RPM inputs during the simulation. However, each mission is stored in a "Missions" directory and can be modified through the MATLAB Command Window prior to running the simulation.

Two existing computer models were used to represent the electric motor in this work. One was developed by Steven Roerig [Ref. 1] for simulating a solar powered vehicle in a thesis at the Naval Post Graduate School. The other is by Krause [Ref. 2: p. 513]. Appropriate modifications have been made and will be explained in this document.

B. THESIS OVERVIEW

At the core of the propulsion model is the electric motor model. Surrounding this motor model is a software feedback system such that the motor is driven to match the torque and RPM

requirements of the mission. The mathematical motor model calculates the necessary voltage and current requirements based on the selected motor parameters. The current values are then used to iterate a battery model that follows empirical discharge curves. To ensure that the motor model accurately represents an actual motor, an empirical efficiency matrix is used to account for various losses in the motor. The resulting simulation provides a means of evaluating various parameters associated with the mission profile and requirements.

The purpose of this thesis is to develop an accurate model of an electric propulsion system that can be used to simulate various missions prior to field testing of UAV hardware systems. This pre-testing will allow the designers to evaluate their design and make modifications before the UAV is actually built. Additionally, substantial effort has been invested in the development of the user interface in hope that ease of use will provide incentive for the model's inclusion in the design and evaluation process.

1. Modeling the Electrical Components

Chapter II covers the basis for the motor and battery models. The mathematics of the permanent-magnet brushless dc motor model is fully covered in Roerig's thesis [Ref. 1]. Therefore, only the developed state space form of the motor model and are discussed. The battery characteristics and motor efficiencies are developed from measured data from the manufacturer. This data is converted to look-up tables which are then used by the system model.

2. Simulation Software and Model Development

MATLAB in conjunction with Simulink, both from MathWorks [Ref. 3], were used as the simulation software because they provide excellent features and are easy to use. The motor model is primarily a set of differential equations and the software facilitates representing these equations in a block diagram form. Familiarity with these tools and the fact that Roerig's model [Ref. 1] also used these tools was an overriding factor in choosing them. The propulsion model is broken up into sub-models which are connected together to form the representation of the

desired function. This structure makes it easier to isolate a specific area for evaluation or modification.

3. Model Testing and Evaluation

Roerig's model and a "simplified" model given by Krause [Ref. 2: p. 513] were evaluated over various input requirements for an Aveox 1817 motor and the results are presented. In the testing process, Krause's model proved more accurate than Roerig's and therefore was chosen as the simulation model.

4. Conclusions and Future Work

Conclusions are made as to the viability of the electric propulsion simulator and a proposal for including the simulator into the design framework is made. Additional ideas are presented as possible follow-on work to enhance the benefit of the simulator.

II. MODELING THE ELECTRICAL COMPONENTS

A. MODEL OVERVIEW

An overview of the electric propulsion model is illustrated in Figure 2-1. The Torque and RPM requirements generated by the aerodynamic model are coupled to the Motor Model. A feedback system forces the motor to match the mission Load Torque and Load RPM. Actual motor losses are accounted for by the Efficiency Matrix. This matrix is constructed from manufacturer's data for the given motor. One of the motor model outputs is the current required

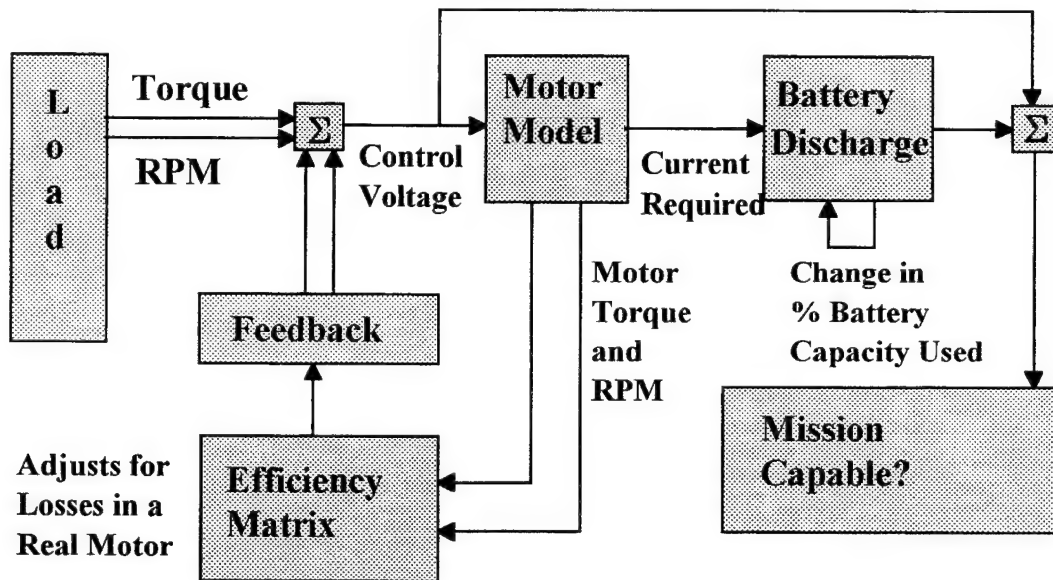


Figure 2-1. Block Diagram of Electric Propulsion Model

by the motor. This current is used as the input to the battery discharge model. For a given state of charge a battery will discharge differently based on the amount of current drawn because of internal losses in the battery. The amount of energy drawn out of a battery is a function of the integral of the current drawn over time. Therefore, a new battery capacity can be found by simply subtracting this current drain from the old battery capacity. For a given percentage of rated capacity and a given current, the battery voltage is identified from the empirical discharge

curves which plot battery voltage against the percent battery capacity used. The battery voltage can then be compared to the voltage required to drive the motor. If the motor voltage exceeds the available battery voltage at any time, this will imply that the maneuver requiring that particular torque and RPM would be impossible to execute for the given motor and battery combination. This discrepancy will indicate to the mission planners or system designers that modifications need to be made. These modifications can take on any of the following forms:

1. An increase in the Amp-Hour Rating of the battery.
2. An increase in the number of battery cells used.
3. Changing the motor parameters.
4. Reducing the Torque and/or RPM requirements of the mission.
5. Some combination of the above.

B. THE PERMANENT-MAGNET BRUSHLESS DIRECT CURRENT (PMBDC) MACHINE

The foundation for the permanent-magnet brushless dc (PMBDC) motor is discussed by Roerig [Ref. 1: pp 10-17]. However, his development has omissions and errors in several equations. These omissions and errors are discussed and corrected in the following paragraphs.

1. The Development of Motor State Variables

Roerig's equation [Ref. 1: p. 14] describing the relationship of the electromagnetic torque T_e to the actual rotor speed ω_r is

$$T_e = (J_r + J_{ml}) \frac{d\omega_r}{dt} + B\omega_r + T_{motload} \quad (2.1)$$

Where:

J_r is the rotor inertia.

J_{ml} is the inertia of the mechanical load connected to the rotor.

B is the damping coefficient due to friction and windage losses.

$T_{motload}$ is the load torque delivered to the motor.

But Krause [Ref. 2: p. 513] gives the relation as

$$\omega_r = \frac{P/2}{J\rho + B_m}(T_e - T_l) \quad (2.2)$$

Where ρ is the notation for the Heaviside operator $\frac{d}{dt}$, P is the number of motor poles and ω_r is the rotor electrical frequency. Manipulating equation (2.2) where $J = (J_r + J_{ml})$, $B = B_m$ and $T_l = T_{motload}$ gives

$$T_e = J\left(\frac{2}{P}\right)\frac{d\omega_r}{dt} + B\left(\frac{2}{P}\right)\omega_r + T_l \quad (2.3)$$

Roerig's equation relating the electromagnetic torque to the motor current i_{qs}^r [Ref. 1: p. 14] is

$$T_e = \frac{P}{2}\lambda_m^r i_{qs}^r \quad (2.4)$$

Where λ_m^r represents the amplitude of the flux linkages as seen from the stator after transformation to the rotor reference frame. But Krause [Ref. 2: p. 503] gives

$$T_e = \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)[\lambda_m^r i_{qs}^r + (L_d - L_q)i_{qs}^r i_{ds}^r] \quad (2.5)$$

Where i_{ds}^r represents the motor current component at right angles to the rotor axis and L_d and L_q represent the inductance along the respective axes. Both Roerig and Krause make the assumption that L_d and L_q are equal (a non-salient rotor). Therefore, equation (2.5) becomes

$$T_e = \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)\lambda_m^r i_{qs}^r \quad (2.6)$$

Combining equations (2.3) and (2.6) gives

$$\left(\frac{3}{2}\right)\left(\frac{P}{2}\right)\lambda_m^r i_{qs}^r = J\left(\frac{2}{P}\right)\frac{d\omega_r}{dt} + B\left(\frac{2}{P}\right)\omega_r + T_l \quad (2.7)$$

Which yields

$$\frac{d\omega_r}{dt} = -\frac{B}{J}\omega_r + \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)^2 \frac{\lambda_m^r}{J} i_{qs}^r - \left(\frac{P}{2}\right) \frac{T_l}{J} \quad (2.8)$$

As opposed to Roerig's [Ref. 1: p. 13]

$$\frac{d\omega_r}{dt} = -\frac{B}{J}\omega_r + \left(\frac{P}{2}\right)^2 \frac{\lambda_m^r}{J} i_{qs}^r - \left(\frac{P}{2}\right) \frac{T_l}{J} \quad (2.9)$$

By substituting:

The electrical time constant $\tau_a = \frac{L_s}{r_a}$ where L_s is defined as equal to L_d and L_q and r_a is the winding resistance.

The back-emf constant $k_e = \lambda_m^r$ and

The torque constant $k_t = \left(\frac{3}{2}\right)\left(\frac{P}{2}\right) \lambda_m^r = \left(\frac{3}{2}\right)\left(\frac{P}{2}\right) k_e$

into his two-phase system [Ref. 1: p. 15] and despite previously discussed errors, Roerig derived with the correct three-phase model represented by

$$\frac{d}{dt} \begin{bmatrix} i_{qs}^r \\ i_{ds}^r \\ \omega_r \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau_a} & 0 & -\frac{k_e}{r_a \tau_a} \\ 0 & -\frac{1}{\tau_a} & 0 \\ \frac{Pk_t}{2J} & 0 & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} i_{qs}^r \\ i_{ds}^r \\ \omega_r \end{bmatrix} + \begin{bmatrix} -\omega_r i_{ds}^r \\ \omega_r i_{qs}^r \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{r_a \tau_a} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{P}{2J} \end{bmatrix} \begin{bmatrix} v_{qs}^r \\ 0 \\ T_l \end{bmatrix} \quad (2.10)$$

where the constants J , P , r_a , τ_a , k_e , and k_t are required model parameters for a given motor and v_{qs}^r represents the voltage component along the rotor axis in the rotor reference frame.

Solving for i_{qs}^r , i_{ds}^r , and ω_r the following differential equations are obtained

$$i_{qs}^r = \left(\frac{1}{r_a}\right) \left(\frac{1}{s\tau_a + 1}\right) (v_{qs}^r - \omega_r k_e - \omega_r i_{ds}^r r_a \tau_a) \quad (2.11)$$

$$i_{ds}^r = \left(\frac{1}{r_a}\right) \left(\frac{1}{s\tau_a + 1}\right) \omega_r i_{qs}^r r_a \tau_a \quad (2.12)$$

$$\omega_r = \left(\frac{P}{2}\right) \left(\frac{1}{sJ + B}\right) (i_{qs}^r k_t - T_l) \quad (2.13)$$

and are used to generate the time-domain block diagram used in the computer model as shown in Figure 2-2.

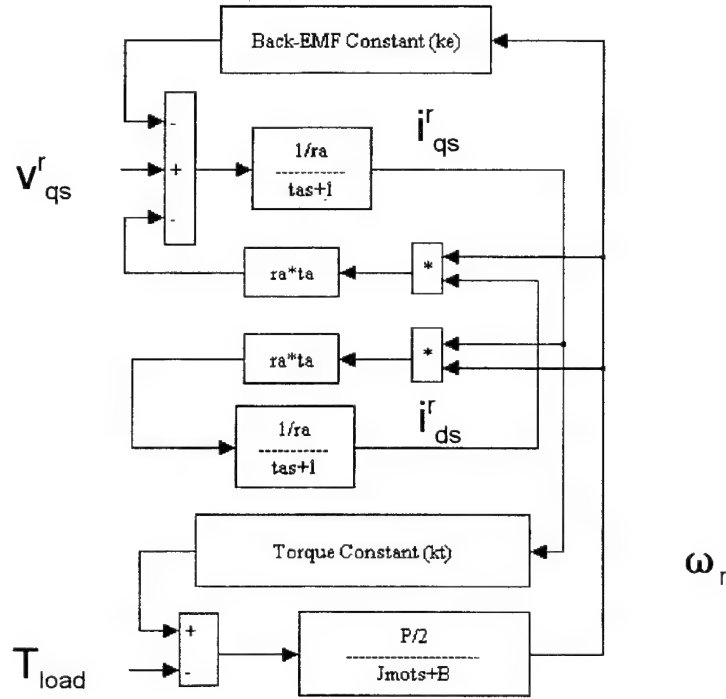


Figure 2-2. Block Diagram of Motor Model Equations

Krause [Ref. 2: p. 512] simplifies this model by substituting equation (2.12) into equation (2.11) which results in

$$i_{qs}^r = \left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a + 1}\right)(v_{qs}^r - \omega_r k_e - \omega_r \left(\left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a + 1}\right)\omega_r i_{qs}^r r_a \tau_a\right) r_a \tau_a) \quad (2.14)$$

Combining like terms gives

$$i_{qs}^r = \left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a + 1}\right)(v_{qs}^r - \omega_r k_e - \left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a + 1}\right)i_{qs}^r (\omega_r r_a \tau_a)^2) \quad (2.15)$$

Since $\tau_a = \frac{L_s}{r_a}$ this becomes

$$i_{qs}^r = \left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a+1}\right)(v_{qs}^r - \omega_r k_e - \left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a+1}\right)i_{qs}^r \omega_r^2 L_s^2) \quad (2.16)$$

In Krause [Ref. 2: p. 512], he then proceeds to neglect the $\omega_r^2 L_s^2$ term. If the same assumption is employed here, equation (2.16) becomes

$$i_{qs}^r = \left(\frac{1}{r_a}\right)\left(\frac{1}{s\tau_a+1}\right)(v_{qs}^r - \omega_r k_e) \quad (2.17)$$

Therefore, the three motor equations (2.11) through (2.13) can be reduced to just two equations: (2.13) and (2.17). Figure 2-3 shows these equations in block diagram form where the simplification is readily apparent as compared to Figure 2-2.

The restrictions that Krause [Ref 2: p 514] places on the simplified model is that v_{qs}^r , ω_r , and T_e should not be negative. Although these restrictions may have a profound

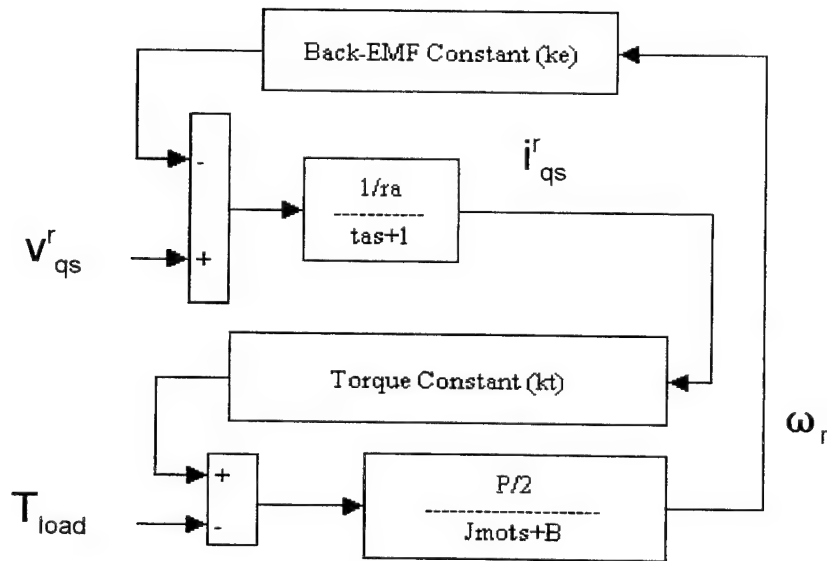


Figure 2-3. Krause's Simplified Model

effect on some systems, they should not pose a problem for a UAV driven by a propeller with a single direction of rotation.

2. Voltage, Current and Power Equations

Another equation in Roerig's work that needs to be corrected is the one relating the input voltage and current to the motor q and d-axis voltage and current. In Krause's book [Ref. 2: p. 488] the power balance between the input of the power converter and the input to the machine is represented as

$$v_i i_i = \frac{3}{2}(v_{qs}^e i_{qs}^e + v_{ds}^e i_{ds}^e) \quad (2.18)$$

where the superscript *e* represents a synchronously rotating reference frame and v_i and i_i are the voltage and current at the inverter input. Since the rotor reference frame for the PMBDC motor is the synchronously rotating reference frame, Krause [Ref. 2: p. 504] expresses the reference frame voltages v_{qs}^r and v_{ds}^r as Fourier series

$$v_{qs}^r = v_{qs}^e = \frac{2v_i}{\pi} \left(1 + \frac{2}{35} \cos 6\varpi_r t - \frac{2}{143} \cos 12\varpi_r t + \dots \right) \quad (2.19)$$

$$v_{ds}^r = v_{ds}^e = \frac{2v_i}{\pi} \left(\frac{12}{35} \sin 6\varpi_r t - \frac{24}{143} \sin 12\varpi_r t + \dots \right) \quad (2.20)$$

Equations (2.19) and (2.20) assume that the power converter produces a six-step voltage output waveform. If the harmonics are ignored, then $v_{qs}^r = \frac{2v_i}{\pi}$ and $v_{ds}^r = 0$. Substituting these values into equation (2.18) gives

$$v_i i_i = \frac{3}{2} v_{qs}^r i_{qs}^r = \frac{3}{2} \left(\frac{2v_i}{\pi} \right) i_{qs}^r \quad (2.21)$$

and then canceling like terms gives

$$i_i = \frac{3}{\pi} i_{qs}^r \quad (2.22)$$

Since the i subscript refers to the inverter input for this motor, V_{dc} and I_{dc} are equal to v_i and i_i respectively. This assumes that the motor uses voltage speed control and 100 percent duty cycle if a Pulse Width Modulation (PWM) inverter is used. Therefore, the equations relating the voltages, currents, and powers are

$$V_{dc} = \frac{\pi}{2} v_{qs}^r \quad (2.23)$$

$$I_{dc} = \frac{3}{\pi} i_{qs}^r \quad (2.24)$$

$$V_{dc} I_{dc} = \frac{3}{2} v_{qs}^r i_{qs}^r \quad (2.25)$$

as opposed to Roerig's [Ref. 1: p. 16] equation

$$V_{dc} I_{dc} = \frac{3}{\pi} (v_{qs}^r i_{qs}^r) \quad (2.26)$$

where he sets $V_{dc} = v_{qs}^r$.

Equations (2.23) through (2.25) are critical to the interaction of the motor with other model components which will be discussed in Chapter III.

3. Accounting for Copper Losses in the Armature Windings

In the model, an efficiency factor is applied to account for the losses associated with an actual motor. This efficiency factor is obtained from an efficiency matrix developed from manufacturer's data. This matrix is an ordered set of the ratio of the motor controller electrical input and the motor mechanical output power at various load torques and motor RPMs. Therefore, the efficiency factor includes all losses occurring within an actual motor and motor controller (resistance, hysteresis, windage, etc.). One of those losses occurs as the motor current passes through the armature windings which have some non-zero resistance. This loss, referred to as "copper loss", is accounted for in the equations used for the motor model. Because both the ideal motor equations and the efficiency factor include this loss it is counted

against the motor twice. Since the efficiency matrix cannot be adjusted to remove the effects of the copper loss, the torque output of the ideal motor is increased by an amount equivalent to the calculated copper loss so that the motor output is truly "ideal" and the efficiency factor can be correctly applied.

The Park's transformation is the starting point to develop the equations necessary for calculating this equivalent torque increase. From Krause [Ref. 2: p. 135]

$$\mathbf{f}_{qd0s} = \mathbf{K}_s \mathbf{f}_{abcs} \quad (2.27)$$

where \mathbf{f} may represent a voltage, current or flux linkage and

$$\mathbf{f}_{abcs} = \mathbf{K}_s^{-1} \mathbf{f}_{qd0s} \quad (2.28)$$

or

$$\begin{bmatrix} f_{as} \\ f_{bs} \\ f_{cs} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 1 \\ \cos(\theta - \frac{2\pi}{3}) & \sin(\theta - \frac{2\pi}{3}) & 1 \\ \cos(\theta + \frac{2\pi}{3}) & \sin(\theta + \frac{2\pi}{3}) & 1 \end{bmatrix} \begin{bmatrix} f_{qs} \\ f_{ds} \\ f_{0s} \end{bmatrix} \quad (2.29)$$

which relates the stationary circuit variables to an arbitrary reference frame. Therefore, a single phase of current in the stationary circuit relates to a synchronously rotating reference frame, where $f_{0s} = 0$, in a wye-connected machine, by

$$i_{as} = i_{qs}^r \cos \theta_r + i_{ds}^r \sin \theta_r + 0 \quad (2.30)$$

which after trig identities becomes

$$i_{as} = \sqrt{(i_{qs}^r)^2 + (i_{ds}^r)^2} \cos[\theta_r - \arctan(\frac{i_{ds}^r}{i_{qs}^r})] \quad (2.31)$$

Since $\sqrt{(i_{qs}^r)^2 + (i_{ds}^r)^2}$ represents the peak value of i_{as} , the squared rms term becomes $(\frac{1}{2})[(i_{qs}^r)^2 + (i_{ds}^r)^2]$. Upon multiplying the squared rms term by the winding resistance r_a , and accounting for the three machine phases, the total power dissipation due to copper losses is given by

$$P_{cu} = (\frac{3}{2})[(i_{qs}^r)^2 + (i_{ds}^r)^2]r_a \quad (2.32)$$

The output power of the motor is given by

$$P_{out} = (\frac{2}{p})\omega_r T_l \quad (2.33)$$

so that an equivalent torque due to copper losses is

$$T_{cu} = (\frac{P}{2})(\frac{P_{cu}}{\omega_r}) \quad (2.34)$$

Once calculated this equivalent torque due to the copper losses can be added to the electromagnetic torque produced by the motor to give an output torque with no losses. Then, this adjusted motor torque can be correctly reduced by the efficiency factor to give a final output torque that matches what a real motor would produce under similar operating conditions.

C. THE BATTERY COMPONENTS

The approach used to model the battery was to make use of empirical battery data that could be accessed during the simulation in conjunction with equations developed from the simple battery model shown in Figure 2-4 where R_{in} is the internal resistance of the battery.

There are two primary effects that needed to be accounted for. The first is how the energy in the battery changes as current is drawn from it. The second is the change in battery voltage.

1. Percent Battery Capacity Used (PBCU) vs Current

It can be seen from Figure 2-4 that the total power developed by the battery is

$$P_{tot} = V_{oc}I(t) \quad (2.35)$$

whereas the total energy expended is

$$E_b = \int P_{tot} dt = \int V_{oc}I(t) dt = CAP_b V_{oc} \quad (2.36)$$

Where CAP_b is the capacity of the battery at some State of Charge (SOC). Equation (2.36) assumes that V_{oc} is constant over the integration interval. The integration interval is the simulation time step. Because this time step is small, the assumption is valid. Therefore, a new energy level in the battery can be found by subtracting the total power delivered within the time interval from the old energy level. By dividing through by V_{oc} this relationship becomes

$$CAP_{new} = CAP_{old} - \int I(t) dt \quad (2.37)$$

This equation is also used to determine how much additional Battery Capacity is needed if the Motor Voltage exceeds the Battery Voltage anytime during the simulation.

The following equation is used to get a value of the Percent Battery Capacity Used (PBCU)

$$PBCU = 100 - 100 \frac{CAP(t)}{CAP(0)} \quad (2.38)$$

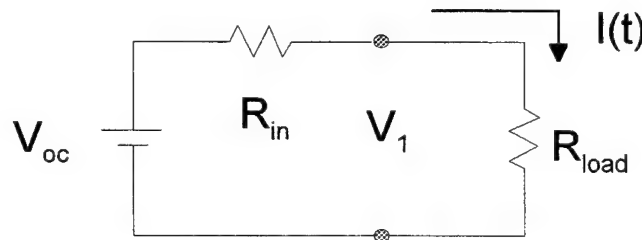


Figure 2-4. Simple Battery Model

2. Battery Voltage vs PBCU and Current

For a given PBCU the battery will produce a certain amount of voltage based on the current required at the time. For this reason, the model accesses a set of battery discharge curves which relate battery voltage, PBCU and current flow for a specific battery type. Figure

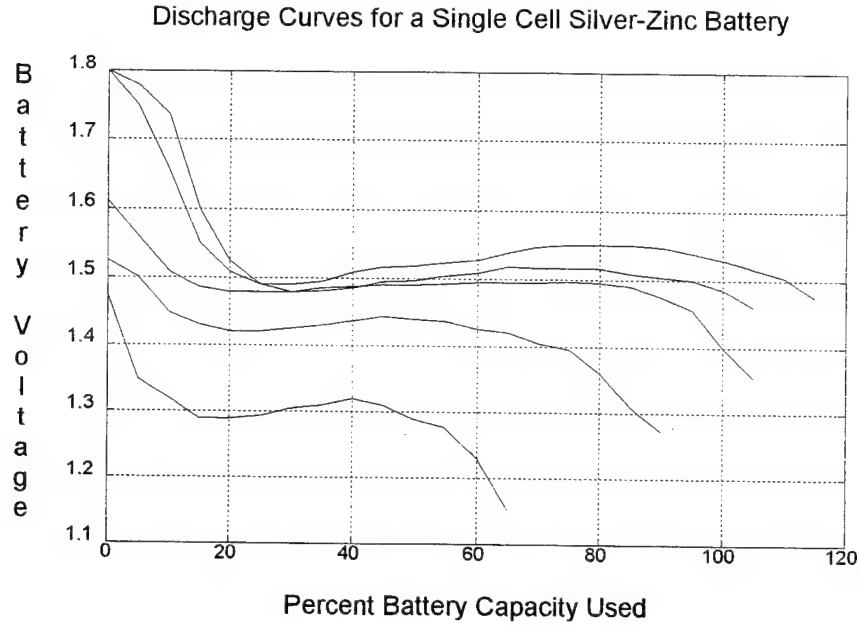


Figure 2-5. Typical Battery Discharge Curves

2-5 shows a set of these curves for a Silver-Zinc cell. These curves are normalized and evaluated for a single battery cell so that they can be adjusted for various battery capacities and numbers of cells in series. Using these curves, the PBCU and the current flow required by the motor, the actual battery voltage is determined. This process is repeated continuously during the simulation to track the available battery voltage.

Battery Efficiency is displayed to the user during the simulation and its value is calculated by

$$\eta_{batt} = 100\left(\frac{V_1}{V_{oc}}\right) \quad (2.39)$$

III. SIMULATION SOFTWARE AND MODEL DEVELOPMENT

A. SIMULATION SOFTWARE

The simulation software used this thesis is MATLAB by Math Works [Ref. 3]. An integral part of the MATLAB package is the Simulink Tool Box. Simulink is the primary tool for building the simulation model whereas the MATLAB code is used for the User Interface. Simulink provides a variety of building blocks whereby block-diagram representations of complex equations and functions can be constructed. Not only is the interface simple to use, but the block diagram approach makes tracing the functions much easier than "sifting" through lines of code. Blocks that represent the individual elements of a function are "dragged and dropped" from the Simulink library to the worksheet. The blocks are then connected as necessary to represent the desired equation or function. Multiple blocks can be grouped together to form new blocks. To open a new block the user needs only to "double click" on it. The input to the model and model variables comes from the MATLAB workspace or from Simulink "Source" functions. The output can be routed back to the MATLAB workspace and is available at the end of the simulation, or can be shown in plots or other Simulink "Sink" functions [Ref. 3].

The User Interface is written in MATLAB code (Appendix B). When the program is activated an Electric Propulsion Simulator (EPS) Control Window is displayed. This control provides various "push buttons" for inputting data pertaining to the simulation and observing the results. Information presented to the user and requests for user input are displayed in the MATLAB Command Window. User instructions are provided in Appendix A.

In Chapter II, the equations for various functions related to the model were shown. Those equations and functions will now be presented in Simulink form.

Figure 3-1 shows the Simulink representation of the complete Electric Propulsion Model. The Motor Control block contains block diagrams representing all the motor equations, the RPM feedback system and the motor efficiency system.

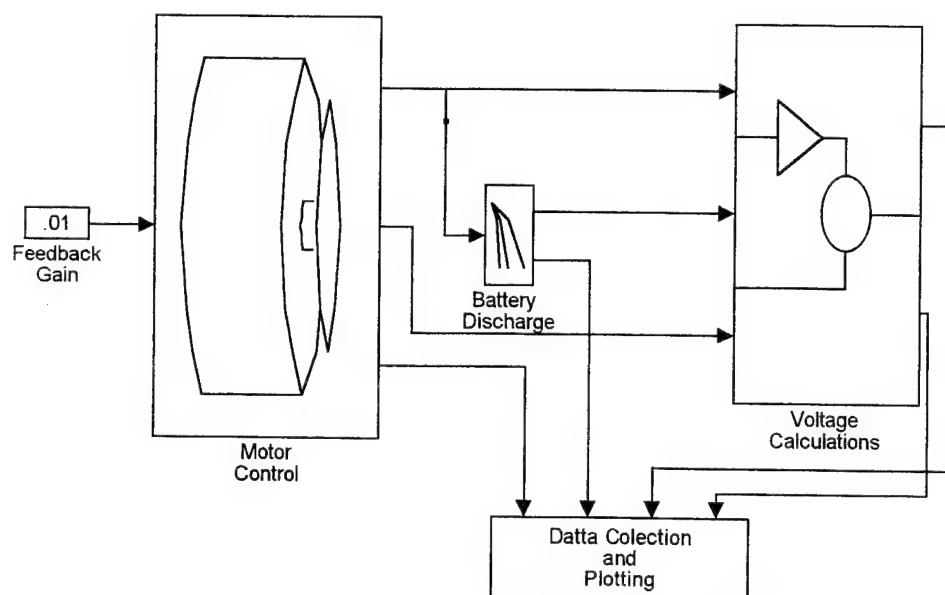


Figure 3-1. Electric Propulsion Model

The Battery Discharge block contains a battery discharge look-up table and necessary functions to calculate the Percent Battery Capacity Used.

In the Voltage Calculations block are the functions related to calculating the items used in summarizing the mission.

Finally, all output data is routed to the Data Collection and Plotting block for display and transfer to the MATLAB workspace.

A feedback gain block is provided so that the user may adjust the amount of gain in the feedback system. An increase in the feedback gain will make the system more responsive to changes in the load requirements but may introduce oscillations. A reduced gain will ensure a smoother response by the system but slow its response to load inputs. Many simulations were conducted and the gain value of .01 seemed to provide the best results. However, the value can be easily changed by double clicking on the block and entering a new value.

B. MOTOR CONTROL BLOCK

Figure 3-2 is obtained when the Motor Control block in Figure 3-1 is "opened" by double clicking on it. Inputs to the Permanent Magnet Brushless Direct Current (PMBDC) Motor are the motor control voltage, the Load Torque and the Motor Efficiency. The model is fed by the Load

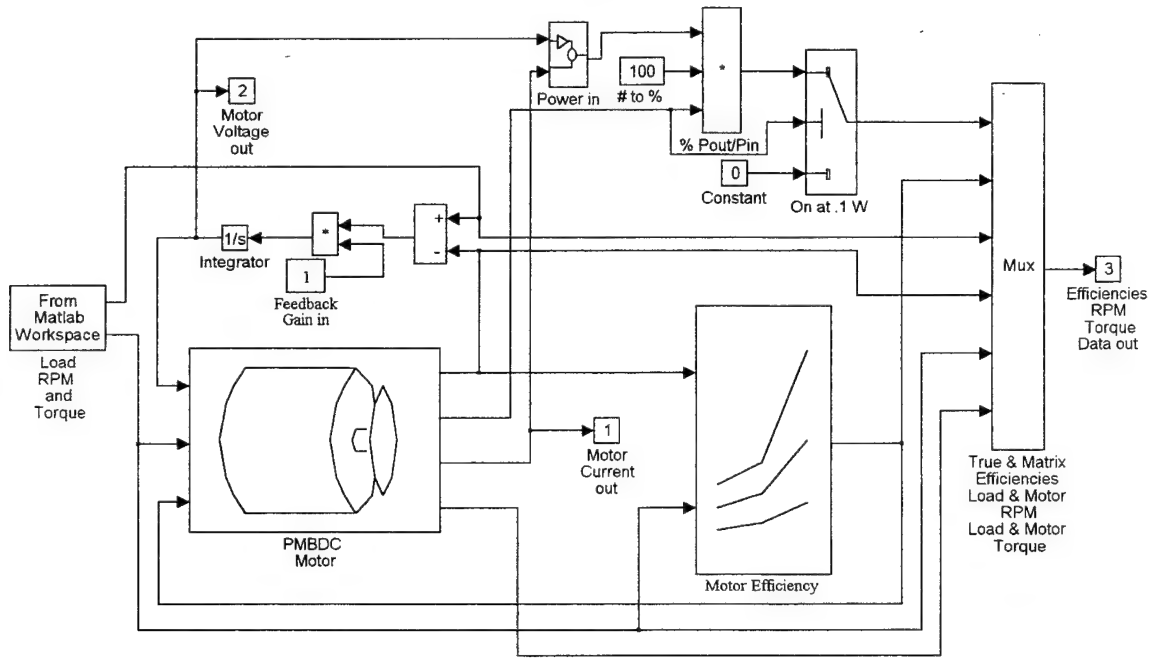


Figure 3-2. Motor Control Block

RPM and Torque block from the MATLAB Workspace. The Load RPM is routed to a simple feedback system that compares the Load RPM with the Motor RPM and integrates the difference to provide a control voltage to the motor. If the Load RPM is greater than the Motor RPM the difference is positive and the integrator responds with an increase in control voltage causing the motor to speed up. If the opposite is true, the integrator will cause a decrease in the control voltage and the motor will slow down. The value of the feedback gain is set by the user as discussed in the previous paragraph. The function of the Load Torque is discussed in the paragraph addressing the PMBDC Motor block.

The outputs of the PMBDC Motor are the Motor RPM, the power delivered by the motor, the Motor Current and the Motor Torque. The power delivered by the motor, in conjunction with the input power to the motor, are used to calculate the actual efficiency of the motor model. This actual efficiency is plotted along with the efficiency generated from the Motor Efficiency look-up table. This plot can be viewed during simulation and provides an indication of whether or not the model is accurately simulating a "real" motor. When plotting, the actual efficiency of the motor model is held at zero until there is enough power delivered to prevent "division by zero" errors. The plotting of calculated values is triggered by the motor output power which must reach a value of .1 Watts.

1. PMBDC Motor Block

Figure 3-3 shows the motor equations (2.11) through (2.13), (2.23) and (2.24) in Simulink form observed when the PMBDC Motor Block is opened. The Load Torque is compared to the Motor Torque after the ideal motor torque is adjusted for "copper losses" and efficiency. The difference in torque is translated to angular acceleration and velocity which is then converted to motor RPM. In this way the model "tracks" the Load Torque requirement by making adjustments in the motor RPM which in turn adjusts the motor voltage through the feedback system discussed earlier. The "modata(3)" term provides a "switch" to ignore the Efficiency Matrix if none exists for the chosen motor. In bypassing the Efficiency Matrix, the model calculates "copper losses" only. Changes to this model by incorporating the simplified Krause model represented by equations (2.13) and (2.17) are discussed at the end of this chapter.

2. Equivalent Torque Block

Contained in the PMBDC Motor block of Figure 3-3 is a sub-block that calculates the torque equivalent of the "copper losses" associated with the ideal motor model. Figure 3-4 shows the contents of that block which is the Simulink representation of equation (2.30). Because of division by the angular velocity (ω_r) the torque equivalent is held at zero until sufficient RPM is developed to prevent "division by zero" errors. This limit is accomplished by a switch that has a threshold set at 100 RPM. The "Memory" block delays the output by one

simulation time step. This function serves to break the algebraic loop formed because the angular velocity, that is an input to the Torque Equivalent Block, is also a function of the output of that same block.

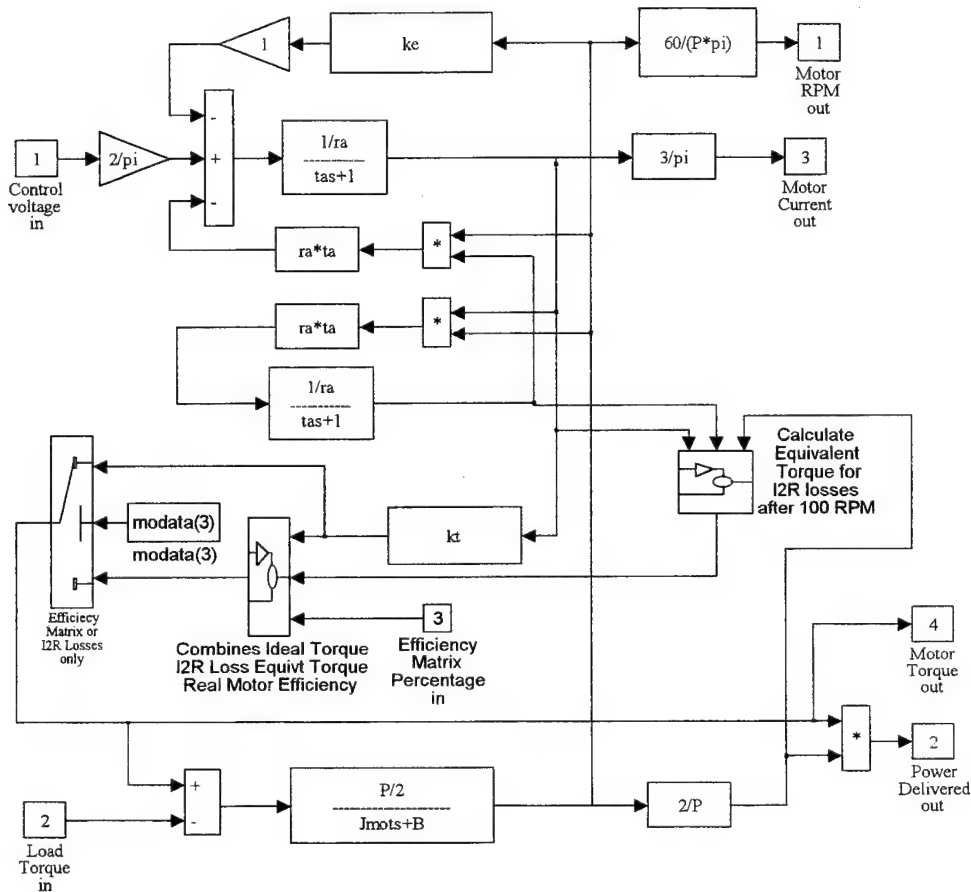


Figure 3-3. PMBDC Motor Block

3. Torque Adjustments Block

Within this sub-block, shown as part of the PMBDC Motor block of Figure 3-3, is the process of converting the "ideal" motor torque to "real" motor torque. The contents of this block are shown in Figure 3-5. Because "copper losses" are accounted for in both the efficiency of a real motor and the equations associated with the motor model, some adjustment needed to be

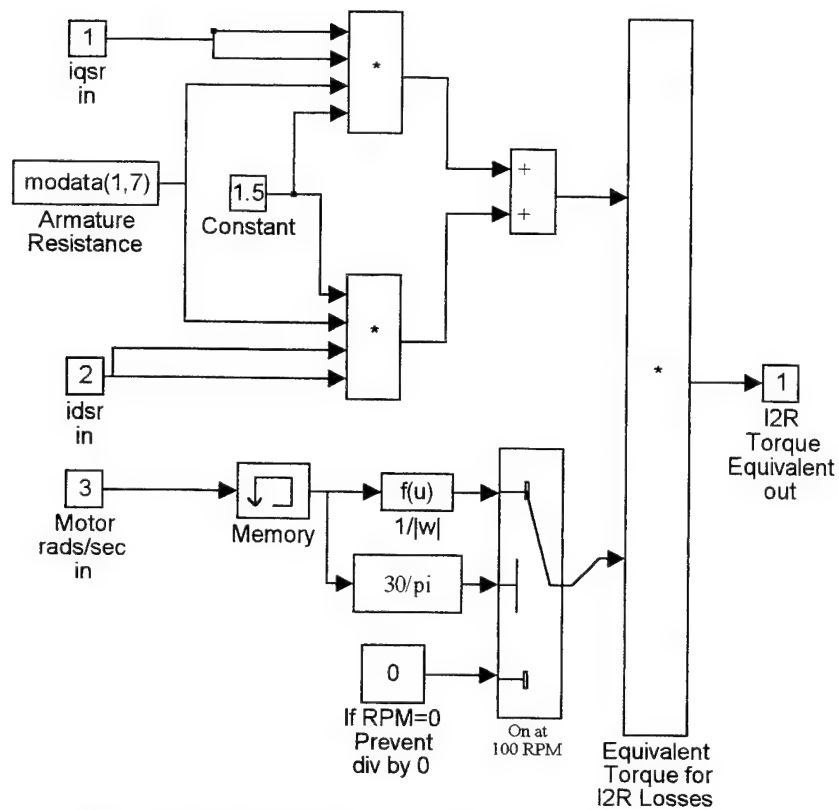


Figure 3-4. Torque Equivalent for Copper Losses

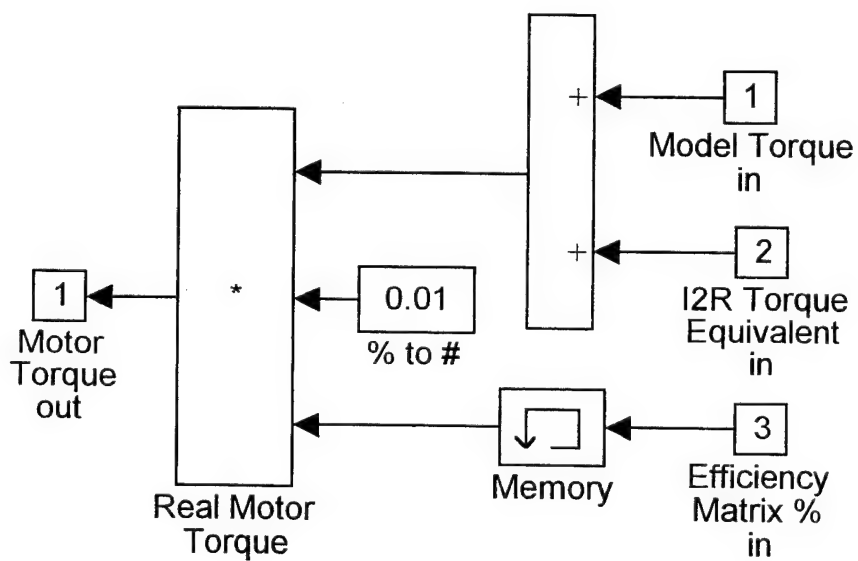


Figure 3-5. Torque Adjustment Block

made in order that this loss is not accounted for twice. In this block, the torque equivalent of the "copper loss" is added to the motor torque. The motor torque now corresponds to an "ideal" motor with no losses. The efficiency factor can now be correctly applied to reduce the torque developed by the model. Therefore, all losses associated with a "real" motor are lumped into an effective torque loss that reduces the "ideal" motor torque in order to "act" like a "real" motor. The "Memory" block serves a similar purpose to that discussed in the previous paragraph.

4. Motor Efficiency Block

The Motor Efficiency block shown in Figure 3-2 is opened up in Figure 3-6. The Efficiency of a motor is a percentage of power out over power in for various torque and RPM

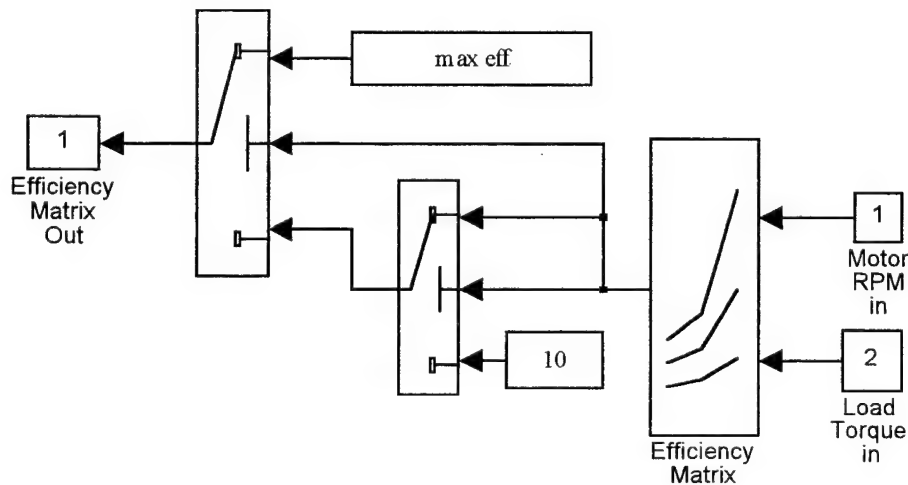


Figure 3-6. Motor Efficiency Block

operating points. Therefore, the Load Torque and Motor RPM are used to access a two dimensional look-up table. The Load Torque is used because the Motor Torque is adjusted by the efficiency and its use produces unacceptable oscillations in the system. The output of the table is the efficiency of a "real" motor when driving a load under the same torque and RPM requirements. The efficiency value then goes through some "logic switches" to ensure that possible extrapolations of the efficiency curves do not produce incorrect results such as

efficiencies in excess of 100 percent. The data for the look-up table is generally specified from motor manufacturer's testing. However, through the User Interface the user can set various efficiencies for design testing if manufacturer's data is not available, or bypass the efficiency factor altogether. The use of other than measured data must be cautioned. The motor model will be forced to operate at the efficiency value obtained from the matrix look-up. Therefore, a constant value efficiency will force the motor model to operate at that efficiency for all operating points even if such operation would be impossible for a "real" motor.

C. BATTERY DISCHARGE BLOCK

Following the flow of simulation, the next block to discuss is the Battery Discharge block which is shown in Figure 3-1 and expanded in Figure 3-7 is discussed next.

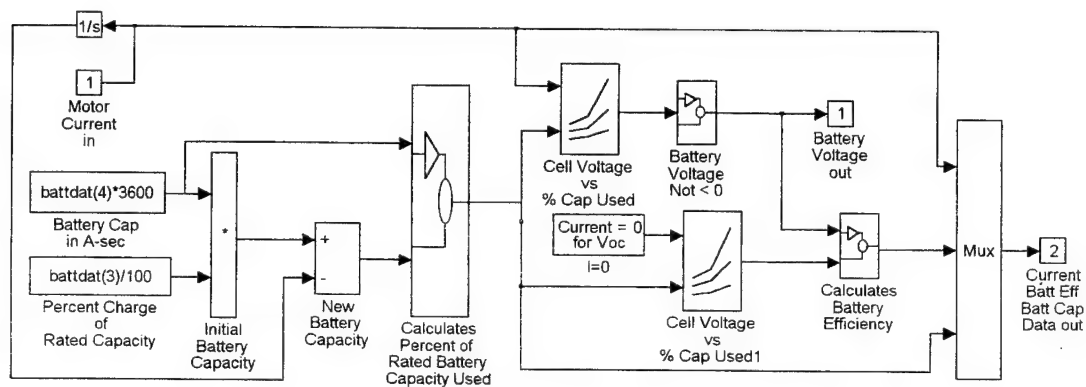


Figure 3-7. Battery Discharge Block

The discharge block takes the initial State of Charge (SOC), the battery rating in Amp-Hours and the Motor Current and iteratively reduces the SOC in accordance with the amount of energy drawn from the battery. Figure 3-7 shows the Simulink representation of the mathematical loop described in equation (2.37). The Cell Voltage two dimensional look-up tables are developed from manufacturer's data as discussed in Chapter II. Interpolation and extrapolation are used to acquire data points not available from manufacturer's data. The Battery Voltage is prevented from being less than zero by the "Battery Voltage Not < 0" block.

Obviously, the better the data the more accurate the look-up will be. The Battery Voltage output is sent to the Voltage Calculations Block for comparison with the Motor Voltage.

D. VOLTAGE CALCULATIONS BLOCK

Shown as part of Figure 3-1, the Voltage Calculations block is expanded and displayed in Figure 3-8.

The purpose of this block is to make calculations and display plots for comparison

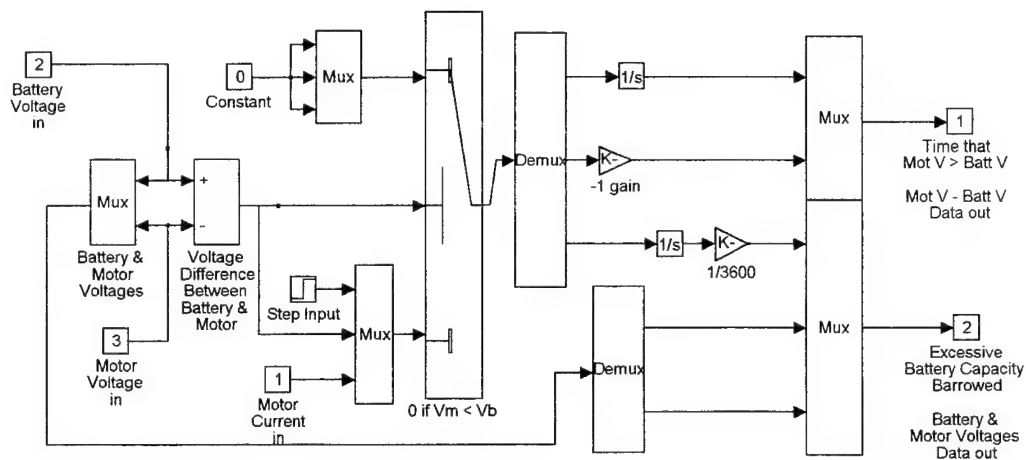


Figure 3-8. Voltage Calculations Block

between the voltages required by the motor and provided by the battery. If the Motor Voltage exceeds the Battery Voltage at any time, the Voltage Calculations block keeps track of the magnitude and duration of such excursions. This information is presented in the Mission Result Summary where corrective recommendations are made. If the Motor Voltage exceeds the maximum Battery Voltage, the user must increase the number of cells in series. If the battery is discharged beyond 80 percent, then additional battery capacity is required. Between these two extremes is the "gray" area that may be corrected by either of the above recommendations or combination of both. The Voltage Calculations block represents these excursions in terms of additional Amp-Hours required.

E. DATA COLLECTION AND PLOTTING BLOCK

As the simulation runs, all output data is displayed in a plot format and/or is transferred to the MATLAB workspace. This data in the workspace is only available after the simulation has been completed or terminated. The data is routed from various generation points to the Data Collection and Plotting Block which is part of Figure 3-1 and opened up in Figure 3-9.

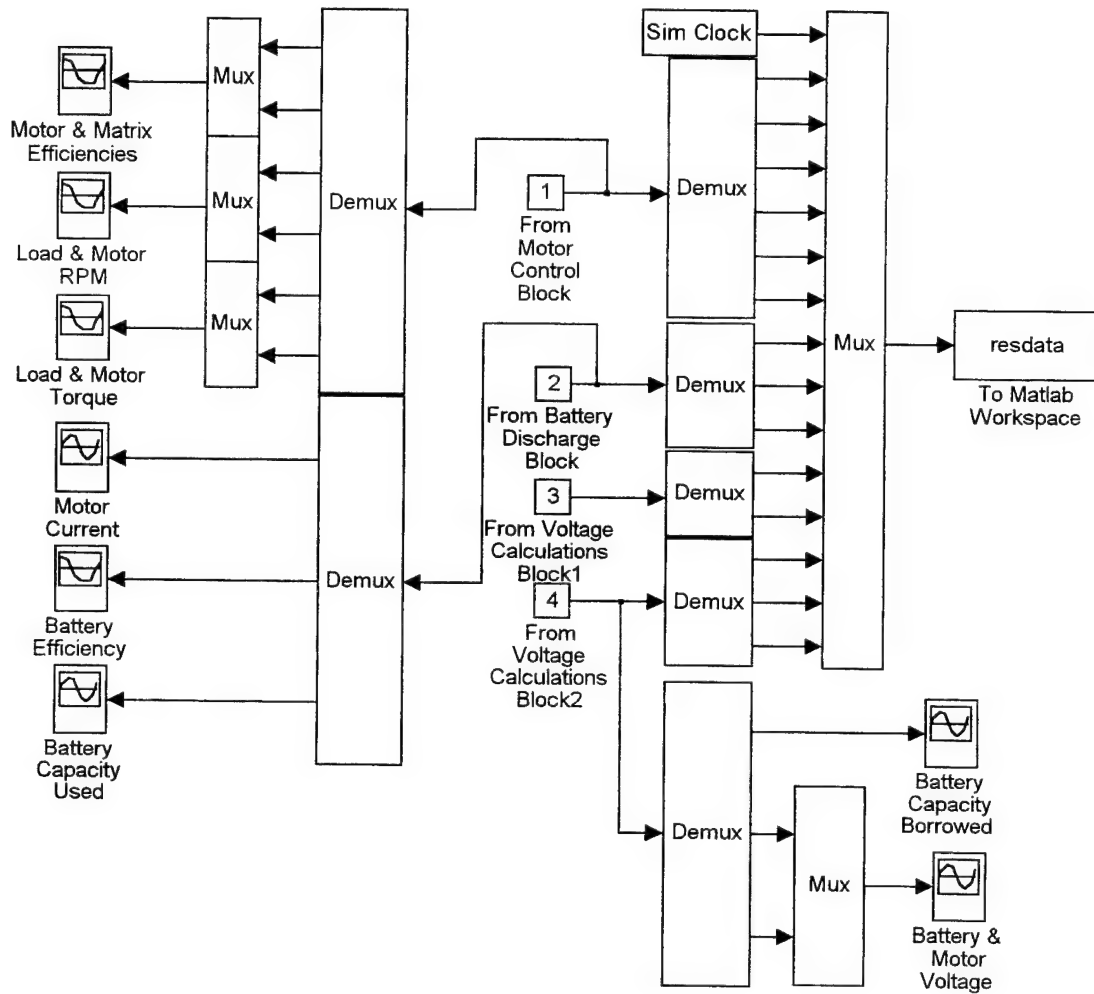


Figure 3-9. Data Collection and Plotting Block

F. SIMPLIFIED KRAUSE MODEL

Incorporating the simplified Krause model results in changes to the PMBDC Motor Block and the Torque Equivalent Block. The altered PMBDC Motor Block is shown in Figure 3-10.

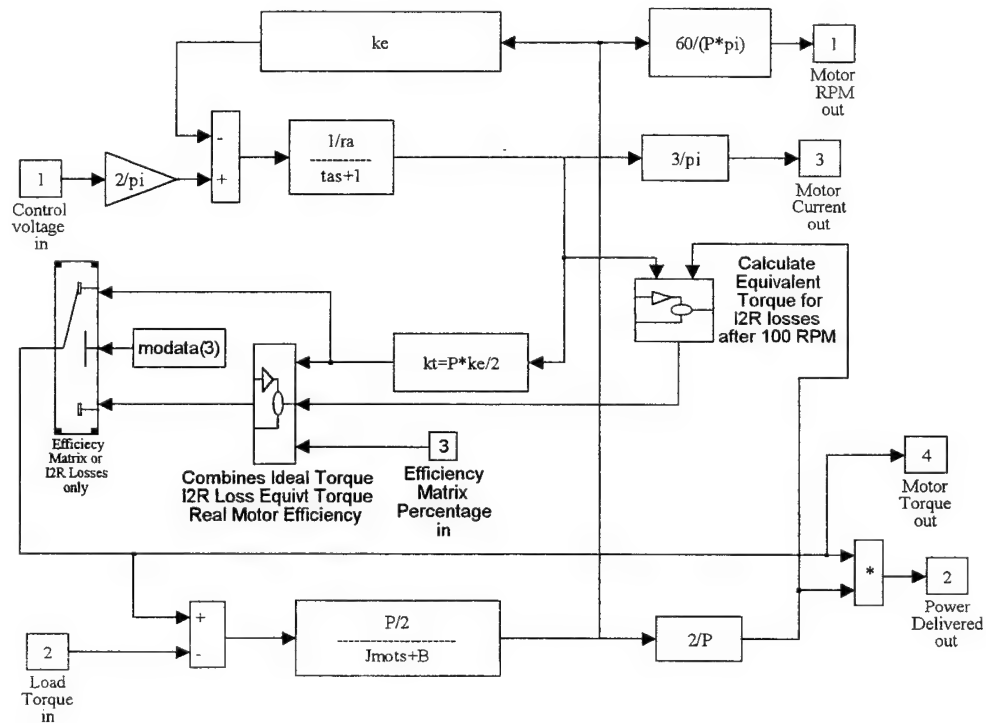


Figure 3-10. Simplified PMBDC Motor Block

Because the simplified model removes the i'_{ds} term, that term is also removed from the "copper loss" calculations as shown in the simplified Torque Equivalent Block of Figure 3-11.

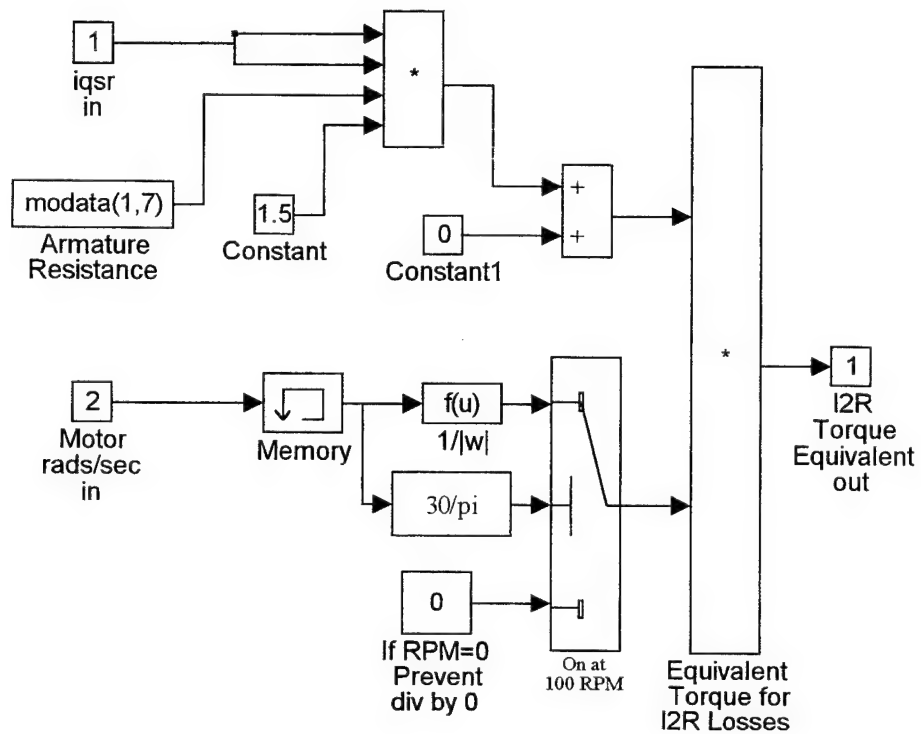


Figure 3-11. Simplified Torque Equivalent for Copper Loss

IV. MODEL TESTING AND EVALUATION

A. OVERVIEW OF THE TESTING APPROACH

To answer the questions of which model to use and how accurate a model is, both models were tested against real-world motor data. This chapter discusses the outcome of the tests and gives an overview of the simulation output.

1. Specification of the Initial Simulation Parameters

The models were compared with a permanent-magnet brushless dc motor built by Aveox. The characteristics of the motor are presented in Table 4-1.

Aveox 1817	
Number of Poles (P)	4
Damping Coefficient (B)	0
Rotor Inertia (J)	1.06e-4 Kg-m ²
Electrical Time-Constant (τ_a)	6.67e-4
Winding Resistance (r_a)	.027 ohms
Back-EMF Constant (k_e) Volts DC	1.538 V/Krpm
Torque Constant (k_t)	.014 N-m/A

Table 4-1. Aveox 1817 Motor Parameters

The Back-EMF Constant in Table 4-1 was given by the manufacturer. However, this constant relates the RPMs to the input DC voltage. The Back-EMF Constant used in the equations of Chapter II is described by Krause [Ref. 3: p. 514] as "the peak phase (winding) voltage." Therefore, in the model, the k_e of Table 4-1 is multiplied by $\frac{2}{\pi}$ to convert the DC

voltage to the phase voltage associated with equation (2.19). Then, the peak voltage k_e is used to calculate k_t .

The Efficiency Matrix for the motor is shown in Figure 4-1. The manufacturer only provided data for 14,000 to 16,000 RPMs. To complete the matrix, the efficiencies at those

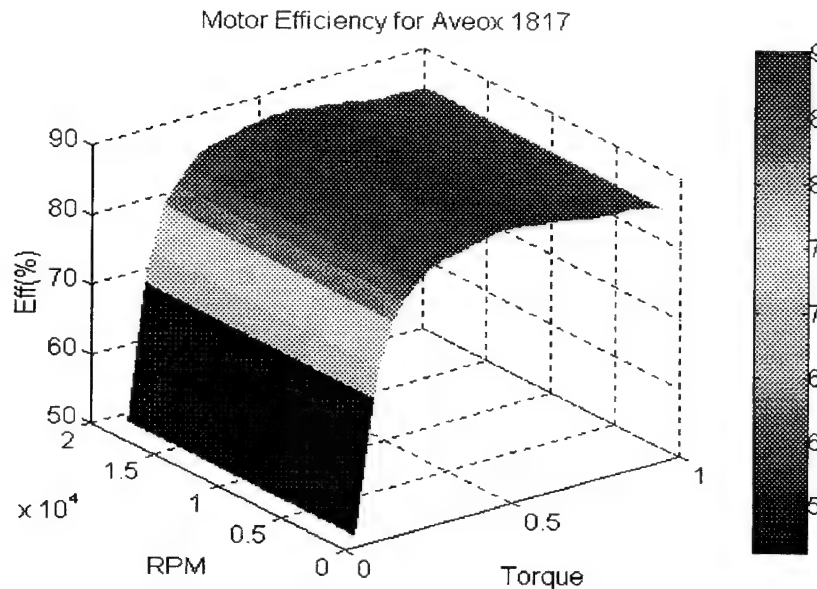


Figure 4-1. Efficiency Matrix for the Aveox 1817 Motor

RPMs were extended over the full range of RPMs. This is obviously not the best method of generating an efficiency matrix, but it may be the best option when limited efficiency data is available. However, the operating points selected for testing were actual points in the manufacturer's data so the efficiency matrix is "exact" for those points.

The battery discharge portion of the simulator was not tested because the simulator battery values come from the manufacturer's own discharge curves (Appendix 3). Currently the available batteries (contained) in the simulator and their manufacturers are:

- | | |
|--|-------------------------|
| 1. Silver-Zinc (Ag-Zn) | Eagle-Picher Industries |
| 2. Nickel-Cadmium (Ni-Cad) | SR Batteries Inc. |
| 3. Lithium-Sulphur-Dioxide (Li-SO ₂) | Pro Battery Specialists |

Figures 4-2 through 4-4 show the battery discharge curves available. The manufacturer's data was extrapolated down to zero volts for use in the simulator. In the battery discharge plots, the letter "C" represents the current value for a one hour discharge. As an example, a 5 Amp-hour battery would have a "C" equal to 5 Amps. C10 would be equal to .5 Amps and 2C would be equal to 10 Amps.

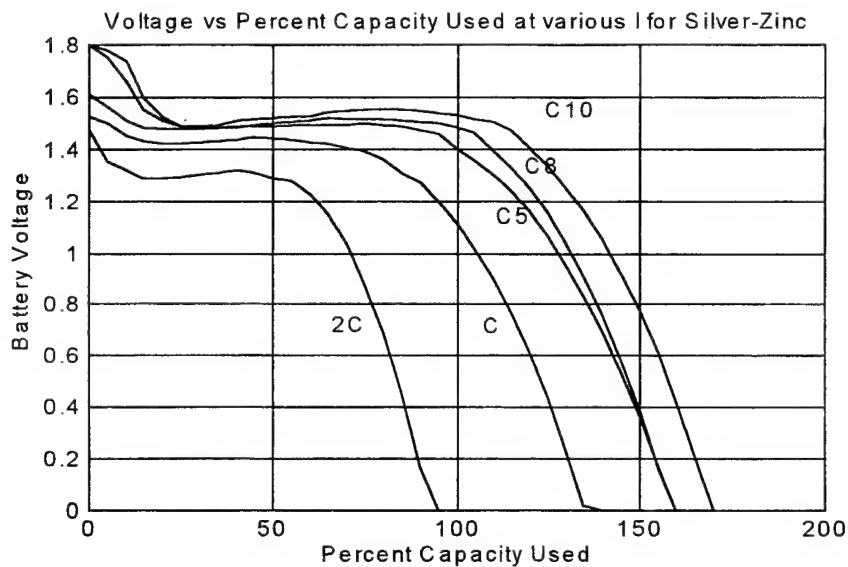


Figure 4-2. Silver-Zinc (Ag-Zn) Single Cell Discharge Curves

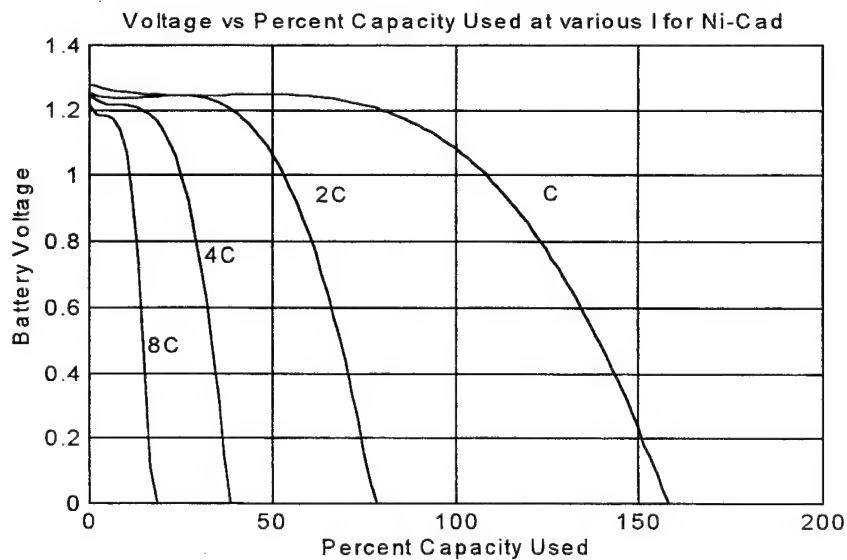


Figure 4-3. Nickel-Cadmium (Ni-Cad) Single Cell Discharge Curves

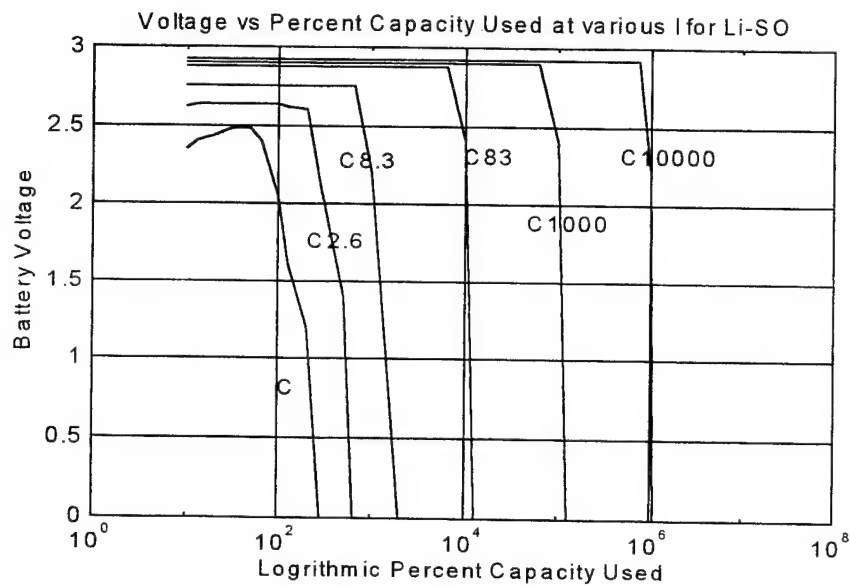


Figure 4-4. Lithium-Sulfur (Li-SO_2) Single Cell Discharge Curves

B. TEST RESULTS

1. Torque and RPM Accuracy

The Electric Propulsion Model was able to track the Load Torque and RPM for both Roerig's and Krause's models. Figure 4-5 shows the Load Torque and the Motor Torque plotted on the same graph for comparison.

Figure 4-6 shows the difference between the Load and Motor Torque. The maximum deviation occurs early in the simulation when the model initially has to accelerate to overcome the inertia of the rotor. The rest of the spikes in the deviations can be attributed to the discontinuities in the Load Torque data.

Figure 4-7 and Figure 4-8 show the comparison of the Load and Motor RPM. These plots indicate that after the initial transients, the model's feedback system is effective in keeping up with the load requirements.

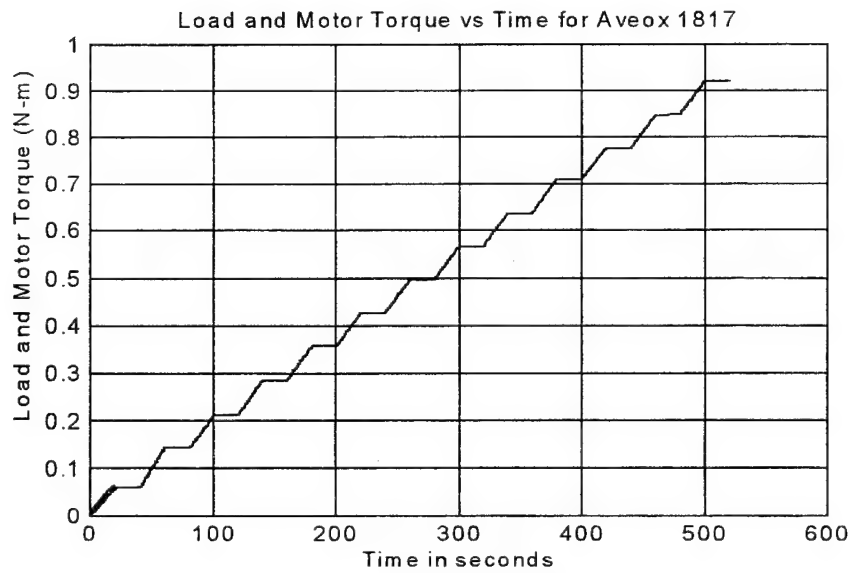


Figure 4-5. Load and Motor Torque

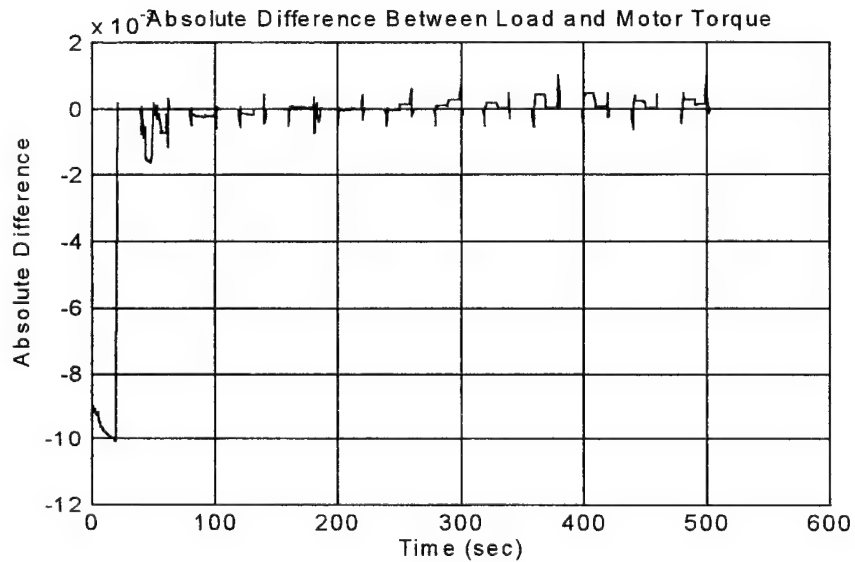


Figure 4-6. Difference between the Load and Motor Torque

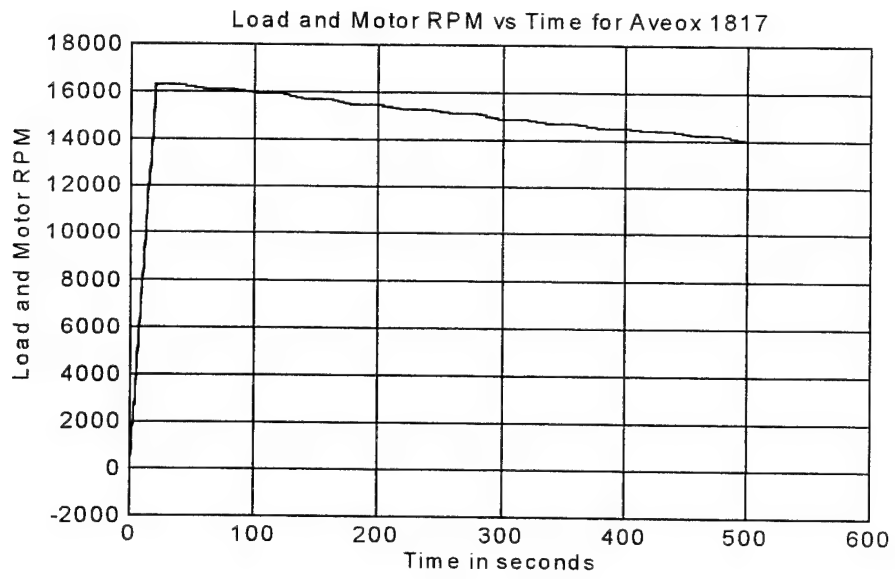


Figure 4-7. Load and Motor RPM

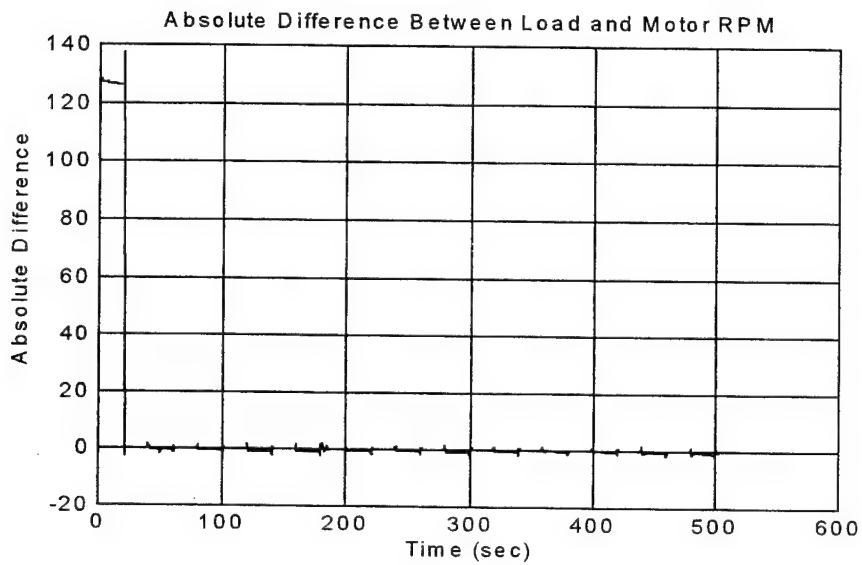


Figure 4-8. Difference Between the Load and Motor RPM

2. Current and Voltage Accuracy

Table 4-2 documents the current and voltage output from Roerig's model when operating at the points indicated as compared to the manufacturer's data. The large Average Percent Difference in current and voltage is why the Krause model was considered.

Torque (N-m)	RPM	Motor Current	Model Current	Percent Difference	Motor Voltage	Model Voltage	Percent Difference
0.057	16,306	6.473	6.115	1.607	23.992	26.76	11.537
0.143	16,140	12.349	10.697	2.618	24.011	27.714	15.422
0.21	15,960	16.98	14.45	17.509	24.139	28.379	17.565
0.284	15,720	22.087	18.397	20.058	24.048	28.935	20.322
0.356	15,480	26.936	22.183	21.426	24.233	29.401	21.326
0.426	15,300	31.607	25.558	23.668	24.175	29.852	23.483
0.5	15,120	36.241	28.888	25.453	24.138	30.262	25.371
0.565	14,880	41.045	32.341	26.913	24.09	30.528	26.725
0.636	14,704	45.755	35.601	22.192	24.023	30.867	28.489
0.71	14,542	50.783	39.104	22.998	24.037	31.27	30.091
0.777	14,400	55.343	42.216	23.719	24.103	31.609	31.141
0.848	14,220	60.186	45.35	24.65	23.949	31.83	32.907
0.921	14,020	65.226	48.535	25.589	23.855	31.987	34.089
Average Percent Difference		Current	19.877		Voltage	24.498	

Table 4-2. Current and Voltage Comparisons of Roerig's Model

Table 4-3 shows the current and voltage output from Krause's model when operating at the points indicated as compared to the manufacturer's data. The low Average Percent Difference makes the simplified model the obvious choice for the Electric Propulsion Simulator.

3. Efficiency Comparisons

Table 4-3 has shown that the model's output is a fairly accurate representation of a "real" motor. Another way to look at how well the model acts like a "real" motor is to compare the efficiency of the manufacturer's data and the efficiency calculated by the model. Using equation (2.33) for the output power P_{out} , the efficiency of the model is simply

$$\eta_{model} = \frac{P_{out}}{P_{in}} = \frac{(\frac{2}{P})\omega_r T_l}{V_{dc} I_{dc}} = \frac{(\frac{2}{P})\omega_r T_m}{V_{dc} I_{dc}} \quad (4.1)$$

where T_{motor} is substituted for T_{load} since these two values are basically equal. Figure 4-9 illustrates the two efficiencies on the same plot.

Torque (N-m)	RPM	Motor Current	Model Current	Percent Difference	Motor Voltage	Model Voltage	Percent Difference
0.057	16,306	6.473	6.451	1.607	23.992	25.365	5.723
0.143	16,140	12.349	11.698	2.618	24.011	25.343	5.547
0.21	15,960	16.98	16.23	4.621	24.139	25.267	4.673
0.284	15,720	22.087	21.192	4.223	24.048	25.119	4.454
0.356	15,480	26.936	26.122	3.116	24.233	24.968	3.033
0.426	15,300	31.607	30.65	3.122	24.175	24.893	2.97
0.5	15,120	36.241	35.553	1.935	24.138	24.819	2.821
0.565	14,880	41.045	40.032	2.53	24.09	24.663	2.379
0.636	14,704	45.755	44.674	2.363	24.023	24.6	2.402
0.71	14,542	50.783	49.755	2.024	24.037	24.575	2.238
0.777	14,400	55.343	54.333	1.825	24.103	24.56	1.896
0.848	14,220	60.186	58.9	2.137	23.949	24.6	2.718
0.921	14,020	65.226	63.654	2.41	23.855	24.39	2.243
Average Percent Difference		Current	2.656		Voltage	3.315	

Table 4-3. Current and Voltage Comparisons of Krause's Model

Because Figure 4-9 gives the impression that the efficiencies are exactly equal, Figure 4-10 is included to show the difference over time. The first 100 points of this plot are omitted because, as discussed in Chapter III, the model efficiency is not calculated until the output power reaches .1 Watt. Because of the transients required between the selected operating points (indicated by the flat line portions of the plots), these areas show the most error. This is the result of the torque and RPM differences between the Load and the Motor as shown in Figures 4-6 and 4-8.

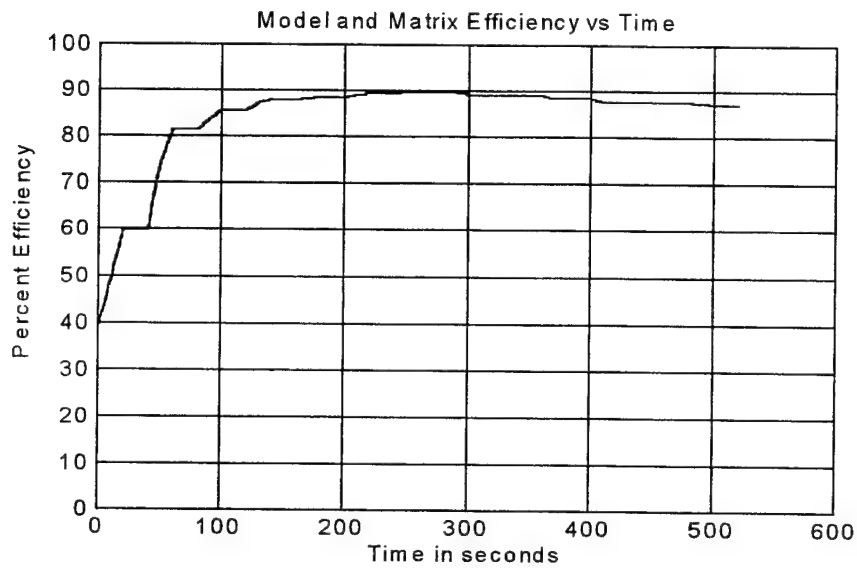
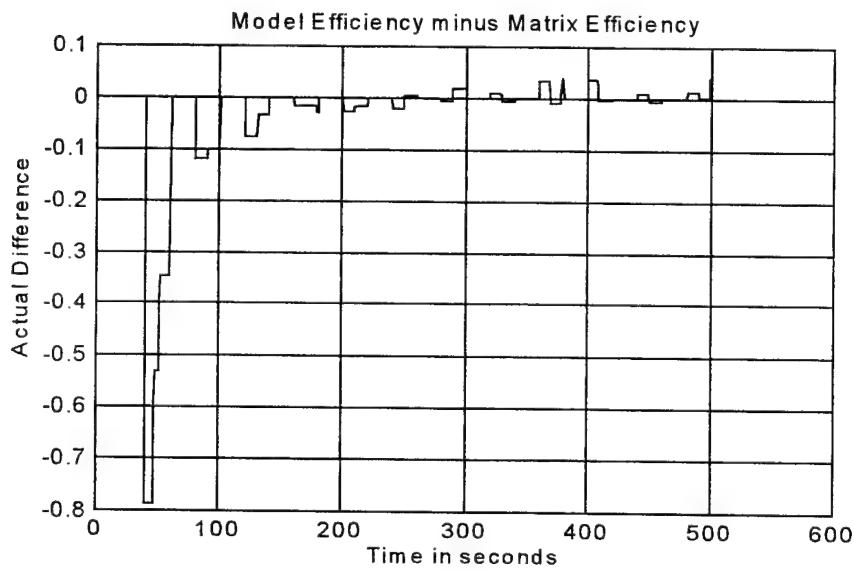


Figure 4-9. Model and Matrix Efficiencies



True
Matrix

Figure 4-10: Difference Between Model and Matrix Efficiencies

C. SIMULATOR OUTPUT

The Electric Propulsion Simulator (EPS) provides output in a variety of plots and MATLAB Command Window information. The following is a list of the plots that can be generated by the EPS.

1. Simulation Plots

◆ Prior to Simulation

1. Load Torque vs Time
2. Load RPM vs Time
3. Nominal Battery Discharge Curves
4. Motor Efficiency Matrix

◆ After Simulation

1. Load and Model Torque
2. Load and Model RPM
3. Model and Matrix Efficiency
4. Model Current
5. Battery and Model Voltage
6. Percent Battery Capacity Used
7. Battery Efficiency
8. Additional Battery Capacity Needed

2. MATLAB Command Window Information

In addition to providing information to the User on requested inputs and operating procedures, the MATLAB Command Window also provides data parameters for the Battery and Motor in use. A listing of these parameters can be found in the Simulator Instructions (Appendix A). Once a simulation has been completed, the User can request a Mission Summary. The following information is contained in that summary.

◆ Standard Information

1. Maximum and Average Motor Current
2. Maximum and Average Motor Voltage
3. Average Motor Efficiency
4. Final Battery Voltage
5. Percent Battery Capacity Used
6. Average Battery Efficiency

- ◆ If Motor Voltage Exceeds Initial Battery Voltage

1. Number of Additional Battery Cells Needed

- ◆ If Motor Voltage Exceeded Battery Voltage and more than 80 percent of the Battery Capacity was used

1. Maximum Difference between Motor and Battery Voltage
2. Average Difference between Motor and Battery Voltage
3. Percentage of Time that the Excursion Occurred
4. Absolute Time of Excursion
5. Additional Battery Capacity Required

- ◆ If Motor Voltage Exceeded Battery Voltage and less than 20 percent of the Battery Capacity was used

1. Maximum Difference between Motor and Battery Voltage
2. Average Difference between Motor and Battery Voltage
3. Percentage of Time that the Excursion Occurred
4. Absolute Time of Excursion
5. Additional Battery Cells Required

- ◆ If Motor Voltage Exceeded Battery Voltage and between 20 and 80 percent of the Battery Capacity was used

1. Maximum Difference between Motor and Battery Voltage
2. Average Difference between Motor and Battery Voltage
3. Percentage of Time that the Excursion Occurred
4. Absolute Time of Excursion

V. CONCLUSIONS

With reduced dollars available for defense spending, there is a need for more computer modeling prior to field testing. The requirement of the model is to be as accurate as possible in order to simulate real-world testing. With the model developed in this thesis the designer can quickly analyze his design for an electric propulsion system and try alternate designs prior to actually building a full-scale model.

This thesis evaluated two possible motor models. The simplified Krause's model was selected because of its improved accuracy in modeling the Aveox 1817 motor. It is unclear why Roerig's model, which is well supported by the equations, failed to produce as accurate results. It may be that the manufacturer's data was misleading in terms of how the measurements were actually taken. This issue is addressed later under Future Work.

A. CURRENT APPLICATION

The goal of this thesis was to develop a model for a UAV electric propulsion system that could assist NRL in the designing and testing of their electrically driven UAVs. That goal has been accomplished with some future work needed. The model performs very well against a limited set of available real-world data. The data that was available did not have measurements of enough operating points to verify accurate operation of the model for all possible modes of operation. Another major issue of concern is the motor parameter constants. Although the terms Torque Constant and Back-EMF Constant are well understood, there is some discrepancy about how they are measured and what they actually represent. The user interface attempts to ensure that the correct constants are used by informing the user of the requirements. This process is described in the Simulator Instructions (Appendix A).

The Electric Propulsion Simulator (EPS) relies on several assumptions about the motors simulated. The model design is based on Permanent-Magnet Brushless Direct Current (PMBDC) Motors. The simulation of other than PMBDC Motors would require modifications to this model.

Those modifications are not dealt with in this thesis. The model also assumes that the motor is controlled by an inverter that uses voltage amplitude speed control only. Other inverter types may use voltage phase or current speed controls. These methods of control are not explored in this thesis.

B. FUTURE WORK

1. Overall System

This model was developed with the constraint that NRL's aerodynamic model would provide the torque and RPM requirements which would drive the model. Obviously, it would be much more efficient to have the aerodynamic model and the electric propulsion model incorporated into a single system.

2. Motor Model

The accuracy of the propulsion model is based on measured data of "real" motors. The Motor Parameters must be measured accurately so that the equations that the model is based on accurately represent that particular motor. Additionally, the ideal nature of the model is converted to a real motor representation by the application of the Efficiency Matrix. Accuracy of simulation results is directly related to the accuracy of this matrix. Therefore, follow on work should include accurate measurements of motor constants and efficiencies over a wide range of operating points.

Additional work could be done in the area of motor control. Various controllers could be modeled which would allow the User to select one to match a particular design type. This additional capability would enable the User to tailor a complete design to his requirements.

3. Battery Model

The battery model is derived from empirical battery discharge data. This assumes a new battery at some nominal temperature. Future work in this area may include extensive battery testing to develop a variety of curves for a variety of conditions (temperature, number of charge/discharge cycles, etc.) that could be accessed by the model. These "Battery Curve

Books" would allow the User to have more control over the specific parameters of a mission that the UAV is required to operate in.

4. Future Applications

Future UAVs could use intelligent "on-board" systems that would incorporate models, like the one represented here, to allow for autonomous or semi-autonomous operation. Such a system could "track" the battery condition and energy usage (which depends on the prevailing meteorological conditions) in flight and select from a variety of possible missions as to what could be accomplished such that there will still be enough battery energy left to return to base safely. With the advantage of autonomous operation and electric power, a UAV would increase its "stealth" capability. Information from and tasking sent to the UAV could be accomplished via burst transmissions, with the UAV itself determining when it needed to return to base.

APPENDIX A. SIMULATOR INSTRUCTIONS

A. STARTING THE ELECTRIC PROPULSION SIMULATOR

1. The Electric Propulsion Simulator (EPS) will only run from a MATLAB Command Window. The MATLAB version must be at least 4.1 with a Simulink version of at least 1.3c. Ensure that these programs are loaded on your computer before attempting to run the simulator.

2. Once in the MATLAB Command Window, type **eps1**. The EPS window will open at the bottom of the screen and necessary default variables will be set. Figure A-1 shows the EPS

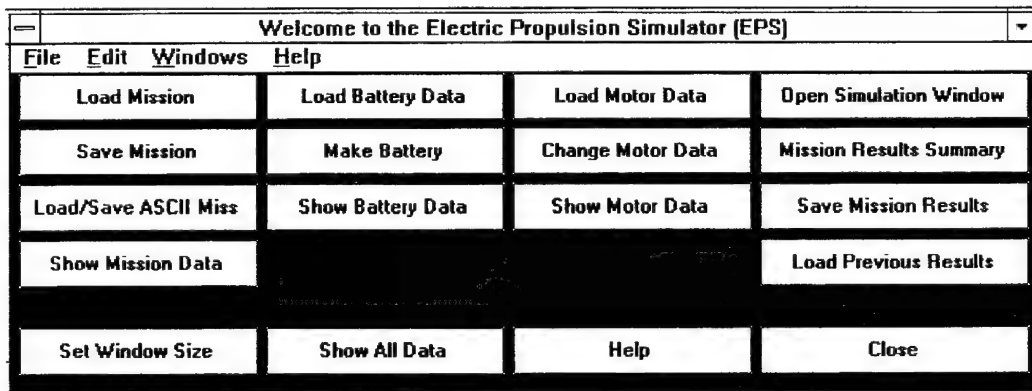


Figure A-1. EPS Control Window

control window. Each button in the EPS Window invokes a specific MATLAB ".m" file. Each of the buttons is explained below.

B. DESCRIPTIONS OF THE EPS CONTROL BUTTONS

1. Load Mission

This button is used to load a set of RPM and Torque Data from the "Missions" directory. A list of available missions in the missions directory will be displayed to the User in the MATLAB command window. The User is requested to select one of the available files or hit Return to

cancel the request. All available files will have a ".run" extension. Once the request is made the data will be loaded into the MATLAB workspace. The data consists of a matrix of three columns of Time, RPM, and Torque, and a mission name.

2. Save Mission

This button allows the User to save a mission into the "Missions" directory. This provision allows the User to make changes to an existing mission and then save it for future use. Mission data is contained in the variable "loadmat" consisting of three columns of Time, RPM and Torque. The User will be asked to name the new mission and provide a file name. The file will then be given the extension ".run".

3. Load/Save ASCII Mission

This button allows the User to convert an ASCII file to MATLAB format and save it to the "Missions" directory. The program will look for any file in the main directory with a three letter extension. The user is requested to choose one of the files. The file is then loaded into the MATLAB workspace. The User is then requested to name the mission and then is asked if the mission is to be saved.

4. Show Mission Data

This button gives the RPM and Torque vs Time plots for the current mission loaded in the MATLAB workspace.

5. Set Window Size

This button displays an outline in the MATLAB Command Window so that the Window can be adjusted to the proper size. This sizing is necessary to ensure that all available information is displayed to the User and not hidden because of automatic scrolling.

6. Load Battery Data

This button allows the User to select the Battery Parameters prior to the simulation start. A file is loaded from the "Battery" directory based on the type of battery desired. The inputs from the User are then used to modify the battery curves. Default parameters have been loaded with the start of the Simulator program. The parameters include:

- ♦ Battery Type
- ♦ Battery Capacity
- ♦ State of Charge
- ♦ Cells in Series

7. Make Battery

This button allows the user to "build" a battery matrix. This is a matrix of single cell battery voltages for a given current and Percent Battery Capacity Used (PBCU). The current index corresponds to the rows of the battery matrix and the PBCU corresponds to the columns. These indexes will be included as the first row and column of the matrix (batcurvs) with index(1,1) as a dummy variable = 999 for a normal plot and 998 for a semi-log plot.

8 Show Battery Data

This button displays the current Battery Parameters currently and generates a plot of the discharge characteristics for the current battery. The values of the discharge current for each curve are displayed on the plot.

9 Show All Data

This button displays all of the currently loaded data. This data includes Mission, Battery and Motor Data. Plots are generated to display the Mission RPM and Torque vs time, Battery discharge curves and the Motor Efficiency, if available.

10 Load Motor Data

This button allows the User to select a motor file from the "Motors" directory. The motor data is then displayed to include:

- ♦ Number of Poles
- ♦ Damping Coefficient
- ♦ Inertia (Kgm^2)
- ♦ Armature Resistance (ohms)
- ♦ Electrical Time-Constant
- ♦ Back-EMF Constant (V/KRPM)
- ♦ Torque Constant (Nm/A)

An Efficiency Matrix is also loaded with the motor file and can be viewed using the "Show Motor Data" button.

11 Change Motor Data

This button allows the user to create a Motor Data file and load the results. This function displays the current motor parameters and prompts the user for the required data to be entered to include:

- ♦ Number of Poles.
- ♦ Damping Coefficient.
- ♦ Inertia (Kgm²).
- ♦ Armature Resistance (ohms).
- ♦ Electrical Time-Constant.
- ♦ Back-EMF Constant (V/KRPM).
- ♦ Torque Constant (Nm/A).

A Return with no entry retains the current motor parameter.

Of particular concern is the Back-EMF Constant. This constant has various definitions but a particular definition is used in the EPS. When entering this parameter, the program will ask if the constant about to be entered relates the KRPM to the D.C. Voltage inverter input or to the Peak Phase Voltage. If the D.C. Voltage choice is selected the entered value will be multiplied by $\frac{2}{\pi}$ to convert the constant to a Peak Phase Voltage per KRPM which is required by the model.

An Efficiency Matrix based on Motor RPM and Load Torque is also required, with the RPM index corresponding to the rows of the Matrix and the Torque index corresponding to the columns. These indexes will be included as the first row and column of the matrix (effmatx) with index(1,1) as a dummy variable = 999. If the Efficiency Matrix is unknown, the user may select from the following defaults.

1. User selected constant value of Efficiency for all values of torque and RPM.
This option must be used with caution since the model will operate at this efficiency which may be impossible for a real motor, thus producing incorrect results.
2. The Efficiency Matrix for the Unique Mobility 127.
3. The Efficiency Matrix for the Unique Mobility 86.

To use these default Efficiency Matrices an RPM and Torque a range must be given to calculate the appropriate indexes.

Additional choices are:

1. To ignore the efficiency matrix all together and just have the model itself calculate and apply the Copper Losses.
2. To retain the currently loaded Efficiency Matrix.

12. Show Motor Data

This button displays the Motor Parameters currently loaded as well as generates the Motor's Efficiency Plot if available.

13. Help

Allows the User to access a Help Menu for each of the EPS buttons. The help given is basically these button descriptions.

14. Open Simulation Window

This button opens up a Simulink Window that contains the EPS Model. Directions concerning the operation of the model are displayed in the MATLAB Command Window.

15. Mission Results Summary

This button provides the User with a summary of the current mission once the simulation has completed or was stopped. The summary information is generated from the output of the simulation, but can only be accessed after the simulation is complete or terminated.

16. Save Mission Results

The User can save all information to a result file in the "Results" directory. Once saved, the data can be retrieved using the EPS "Load Previous Results" button. A new default load can also be saved to the "Missions" directory. It will load each time the EPS is started.

17. Load Previous Results

This button allows the User to load saved results from a previous mission and to display all the plots (10) associated with that mission. These ten plots are as follows:

1. Nominal Battery Discharge Curves
2. Motor Efficiency Matrix
3. Load and Model Torque
4. Load and Model RPM
5. Model and Matrix Efficiency
6. Model Current
7. Battery and Model Voltage
8. Percent Battery Capacity Used
9. Battery Efficiency
10. Additional Battery Capacity Needed

This feature prevents the User from having to run redundant missions to reproduce the same results.

18. Close

This button closes the EPS Window.

C. STEP SEQUENCE IN OPERATING THE EPS

Although the buttons are independent of one another, they operate on the currently loaded data. Therefore, once the EPS Window is opened any of the buttons may be used since default values have already been loaded. The following steps are given to ensure that the user loads all the desired data prior to beginning the simulation.

1. Step One

Use the "Set Window Size" button to adjust your MATLAB Command Window. The Command Window should be about the same width as the EPS Window and at least twice as tall.

2. Step Two

Use the "Load Mission" or "Load/Save ASCII Miss" buttons to load the appropriate mission data. The "Load Mission" button looks for mission files in the "Missions" directory, whereas the ASCII button is used to load an ASCII file into the MATLAB Workspace from the main directory so that it can be saved into the "Missions" directory if desired.

The mission data can be seen in plot form by using the "Show Mission Data" button.

3. Step Three

Use the "Load Battery Data" to select a battery type and enter in the required battery parameters. The battery parameters are displayed and the discharge curves plotted when the "Show Battery Data" button is used.

4. Step Four

Use the "Load Motor Data" to select a motor file from the "Motors" directory. The motor parameters can be changed and saved under a new name by using the "Change Motor Data"

button. The efficiency plot for the currently loaded motor can be seen by using the "Show Motor Data" button.

5. Step Five

Use the "Open Simulation Window" button to access the Simulink Control Window for the Electric Propulsion Simulator. Instructions concerning this window will be displayed in the MATLAB Command Window.

6. Step Six

Once the simulation completes or is terminated, use the "Mission Results Summary" button to display summary information in the MATLAB Control Window. The "Save Mission Results" button can be used to save all of the result data to the "Results" directory.

D. CRITICAL VARIABLES DESCRIPTION

1. loaddat

A three column matrix of Time, Load RPM and Load Torque for each time step.

2. runstr

A string variable that is used as the mission's name.

3. batstr

A string variable that is used to carry the name of the battery type.

4. battdat

A vector of four battery parameter values.

- ♦ battdat(1): Battery Capacity (Amp-Hrs)
- ♦ battdat(2): Initial State of Charge (Percent)
- ♦ battdat(3): Not used - set to a value of one
- ♦ battdat(4): Number of Cells in Series

5. batcurvs

A matrix containing the battery voltage for a given Percent Battery Capacity Used (PBCU) and discharge current. Indexes of PBCU corresponding to the columns of the matrix is included in "batcurvs" as the first row. Indexes of discharge currents corresponding to the rows

of the matrix are included in "batcurvs" as the first column. The dummy value of "batcurvs(1,1)" is used to determine if the discharge curves are plotted on a semilog plot or not.

6. mostr

A string variable containing the name of the motor.

7. modata

A vector of ten values that carry the motor parameters.

- ♦ modata(1): Minimum value of the Efficiency Matrix
- ♦ modata(2): Maximum value of the Efficiency Matrix
- ♦ modata(3): Normally -1, set to 1 if no Efficiency Matrix is available
- ♦ modata(4): Number of Poles in the motor
- ♦ modata(5): Damping Coefficient, normally set to zero
- ♦ modata(6): Rotor Inertial in kg-m²
- ♦ modata(7): Winding Resistance in ohms
- ♦ modata(8): Electric Time Constant
- ♦ modata(9): Back-EMF Constant in V/KRPM
- ♦ modata(10): Torque Constant in N-m/Amp

8. effmatx

A matrix containing the motor efficiency for a given Torque and RPM. Indexes of Torque values corresponding to the columns is included in the matrix as the first row. Indexes of RPM values corresponding to the rows is included in the matrix as the first column. A dummy value of 999 is used to hold the space at "effmatx(1,1)".

9. resdata

After completing its run or when stopped by the operator, the simulator will transfer its output data into the MATLAB Workspace matrix variable "resdata". This matrix contains 15 columns with the following data.

1. Simulation Time
2. True Motor Model Efficiency calculated from Power out over Power in
3. Efficiency Matrix values used during the simulation
4. Load RPM
5. Motor RPM
6. Load Torque
7. Motor Torque
8. Motor Current
9. Battery Efficiency calculated from the Loaded Battery Voltage over the No-load Battery Voltage
10. Percent Battery Capacity Used
11. Time when Motor Voltage exceeded Battery Voltage

12. Voltage difference when Motor Voltage exceeded Battery Voltage
13. Additional Battery Capacity needed
14. Battery Voltage
15. Motor Voltage

APPENDIX B. MATLAB PROGRAMS

A. EPS1.M STARTS EPS

```
%EPS1
%Used to start the Electric Propulsion simulator.
uav5
```

B. UAV5.M EPS USER INTERFACE

```
% Electric Propulsion Simulator
% Masters Thesis EE Naval-Post Graduate School, Monterey
% This system will simulate an electric propulsion system.
%
% This system may be used to optimize vehicle configurations.
% each function is explained separately...
%
% This function initiates the system user control window
%and loads default data.
%
% This system was developed by Steven Roerig, LT, USN
% Date: 5 Mar 1995
%
% and modified by Joel Yourkowski, Maj, USMC
% Date: 23 Feb 1996
%
% *****

% Initialize variables and
% load default data from the "Missions" directory.

resdata = zeros(15);
cd Missions;
load deflt22;
cd ..

% *****
% Initialize the user control window and set the position
%
fignumber=figure( ...
'Position',[5 5 645 200], ...
'Color',[0 0 1], ...
'NumberTitle','off', ...
'Name','Welcome to the Electric Propulsion Simulator (EPS)', ...
'Resize','off', ...
'Pointer','arrow');

% *****
% Display Welcome Message

clc
disp(' ')
disp('*** Welcome to the Electric Propulsion Simulator (EPS). ')
disp(' ')
```

```

disp(' Default data has been loaded to initialize')
disp('all variables.')
disp(' ')
disp(' Please make your selections from the Control Buttons')
disp('in the EPS window and follow the directions given ')
disp('in the Matlab Command window.')
disp(' ')
disp(' Prompts will be given for the next logical step,')
disp('however, selections can be made in any order since')
disp('default values have been loaded.')
disp(' ')
disp('STEP 1: Use the "Set Window Size"')
disp('button to repeat this message and display the size')
disp('of the Matlab Command Window necessary in')
disp('order to see all of the information provided.')
disp(' ')
disp(' At any time use the EPS "Show All Data"')
disp('button to display all currently loaded data')
disp(' ')
disp('STEP 2: Using the EPS "Load Mission" button, load a')
disp('mission to be simulated.')

```

```

% *****
% Create the WINDOW SIZE Button

```

```

sizebut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[5 10 150 30], ...
'Visible','off', ...
'String','Set Window Size', ...
'Enable','off', ...
'Callback','winsize');

```

```

% *****
% Create the HELP Button

```

```

helpbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[315 10 150 30], ...
'Visible','off', ...
'String','Help', ...
'Enable','off', ...
'Callback','helper');

```

```

% *****
% Create the CLOSE Button ... closes the user control window

```

```

closebut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...

```

```
'Position',[470 10 170 30], ...
'Visible','off', ...
'String','Close', ...
'Enable','off', ...
'Callback','close(gcf)');
```

```
% *****
% Create all User Data Buttons
```

```
% *****
% LOAD RUN ... Allows user to load a specific mission from
% the 'missions' directory.
%
runbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[5 170 150 30], ...
'Visible','off', ...
'String','Load Mission', ...
'Enable','off', ...
'Callback','[loadat,runstr]=loadrun(runstr);');
```

```
% *****
% SAVE RUN ... Allows user to save a specific mission from
% the MATLAB workspace to the 'missions' directory.
%
svrunbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[5 135 150 30], ...
'Visible','off', ...
'String','Save Mission', ...
'Enable','off', ...
'Callback','[runstr]=saverun(loadat);');
```

```
% *****
% CONVERT ASCII RUN ... Loads a ASCII mission file and converts it to
% a normal MATLAB file which is then stored in the 'missions' directory.
%
setrunbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[5 100 150 30], ...
'Visible','off', ...
'String','Load/Save ASCII Miss', ...
'Enable','off', ...
'Callback','[loadat,runstr]=ascrun(runstr);');
```

```
% *****
% SHOW MISSION DATA ... Allows the user to view the RPM and Torque
```



```

% requirements of the currently selected mission.
%
shrunbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[5 65 150 30], ...
'Visible','off', ...
'String','Show Mission Data', ...
'Enable','off', ...
'Callback','showdat2(loadaddat,runstr,modata,mostr,effmatx,battdat,batstr,batcurvs,1);');

% *****
% BATTERY DATA ... Allows the user to choose the battery type and
% set Battery Parameters necessary for the simulation
%
battbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[160 170 150 30], ...
'Visible','off', ...
'String','Load Battery Data', ...
'Enable','off', ...
'Callback','[batcurvs,battdat,batstr]=getbatt1(battdat);');

% *****
% MAKE BATTERY ... allows the user to input the Battery
% Data and save it as a file in the "Batterys" directory.
%
mkbattbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[160 135 150 30], ...
'Visible','off', ...
'String','Make Battery', ...
'Enable','off', ...
'Callback','[batcurvs,batstr]=makebat;');

% *****
% SHOW BATTERY DATA ... allows the user to view the Battery
% Data and battery plot for the battery currently in use.
%
swbatbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[160 100 150 30], ...
'Visible','off', ...
'String','Show Battery Data', ...
'Enable','off', ...
'Callback','showdat2(loadaddat,runstr,modata,mostr,effmatx,battdat,batstr,batcurvs,3);');

```

```
% *****
%  SHOW ALL INPUT DATA ... shows all significant data into simulator.
%  Useful after making changes.
%
showbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[160 10 150 30], ...
'Visible','off', ...
'String','Show All Data', ...
'Enable','off', ...
'Callback','showdat2(loaddat,runstr,modata,mostr,effmatx,battdat,batstr,batcurvs,0);');
```

```
% *****
%  GET MOTOR DATA ... allows the user to load a motor file
%  from the directory. This file includes the Motor
%  Parameters and the associated Efficiency Matrix.
%
getmobut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[315 170 150 30], ...
'Visible','off', ...
'String','Load Motor Data', ...
'Enable','off', ...
'Callback','[modata,mostr,effmatx]=getmo(modata,mostr);');
```

```
% *****
%  Change MOTOR DATA ... allows the user to change the Motor
%  Parameters and the associated Efficiency Matrix. Once
%  the data has been changed the user can rename the motor
%  and save it as a particular motor file.
%
setmobut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[315 135 150 30], ...
'Visible','off', ...
'String','Change Motor Data', ...
'Enable','off', ...
'Callback','[modata,mostr,effmatx]=setmo2(modata,effmatx);');
```

```
% *****
%  SHOW MOTOR DATA ... Displays the data for the current motor including
%  the 3-dimensional plot of motor efficiency.
%  Each motor must have an efficiency matrix.
%
effbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[315 100 150 30], ...
'Visible','off', ...
```

```

'String','Show Motor Data', ...
'Enable','off', ...
'Callback','showdat2(loadaddat,runstr,modata,mostr,effmatx,battdat,batstr,batcurvs,2);');

% *****
% Create the RUN Button
%
simbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[470 170 170 30], ...
'String','Open Simulation Window', ...
'Visible','off', ...
'Enable','off', ...
'Callback','openmod');

% *****
% MISSION RESULT SUMMARY ... After the mission, selecting this button
% provides the user with pertinent information on the results of the mission.
%
missumbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[470 135 170 30], ...
'Visible','off', ...
'String','Mission Results Summary', ...
'Enable','off', ...
'Callback','misssum(battdat,resdata);');

% *****
% SAVE MISSION RESULTS ... After the mission is run, all data
% from the mission
% can be saved in the 'results' directory.
%
savmisbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[470 100 170 30], ...
'Visible','off', ...
'String','Save Mission Results', ...
'Enable','off', ...
'Callback','misssave(loadaddat,runstr,batstr,battdat,batcurvs,mostr,modata,effmatx,resdata);');

% *****
% LOAD PREVIOUS MISSION RESULTS ... Load the results of a previously run
% mission from the 'results' directory.
%
loadmisbut = uicontrol( ...
'Style','pushbutton', ...
'Units','pixels', ...
'Position',[470 65 170 30], ...
'Visible','off', ...

```

```
'String','Load Previous Results', ...
'Enable','off', ...
'Callback','[loadat,runstr,batstr,battdat,batcurvs,mostr,modata,effmatx,resdata]=missload;');
```

```
% *****
% Turns on all buttons ....
%
```

```
set([runbut svrunbut shrunbut setrunbut battbut mkbatbut swbatbut getmobut ...
    setmobut effbut showbut simbut missumbut savmisbut loadmisbut ...
    sizebut helpbut closebut], ...
    'Enable','on','Visible','on');
```

C. LOADRUN.M LOADS MISSION FROM "MISSIONS" DIRECTORY

```
function [loadat,runstr]=loadrun(runstr);
%LOADRUN [loadat,runstr] = loadrun(runstr)
% Load mission data from selected '*.run' file in
%'Missions' directory. This function lists the available
%*.run' files in the 'Missions' directory and tasks the
%user to chose one. The chosen file is loaded and the
%data can be displayed using the SHOWRUN button.
%
%loadat: Variable containing the mission data in the form
% of three columns: Time(sec) RPM Torque(N-m)
%runstr: Variable containing a description of the mission.
%

cd Missions
clc;pause(.001)
try_again = 1;
while try_again ==1
    disp(' ')
    disp('*** The following Mission Files are available')
    disp(' ')
    dir *.run;
    disp(' ')
    disp('*** Chose a Mission File, use no extension; (.run assumed)')
    xrun = input('Or Return to cancel request: ','s');
    xrun1 = [xrun,'.run'];

    if (exist(xrun1) == 2)
        eval(['load ',xrun1,' -mat']);
        try_again = 0;
        clc;pause(.001)
        disp([' ']);
        disp(['*** ',xrun1,' Loaded; Mission Name: ',runstr])
        disp(' ')
        disp(' Mission RPM and Torque Plots can be seen by')
        disp('using the EPS "Show Mission Data" button.')
        disp(' ')
        disp(' STEP 3: Using the EPS "Load Battery Data"')
        disp('button, load a selected battery.')
```

```

elseif isempty(xrun) == 1
    try_again = 0;
    clc; pause(.001)
    disp(' ')
    disp('*** Request Canceled')
    disp('Current Mission Data is retained')
    disp(['Mission Name: ', runstr])

else
    disp(' ')
    disp('*****')
    disp(['*** NO SUCH FILE AS: ', xrun1, '!'])
    disp('*****')
end
end
end
cd ..

```

D. SAVERUN.M SAVES MISSION TO "MISSIONS" DIRECTORY

```

function [runstr] = saverun(loadat);
%SAVERUN [runstr] = saverun(loadat)
% Save mission data to selected '*.run' file in 'Missions'
% directory. This function allows the user to alter the
% existing mission and save it to a new file or overwrite
% one.
%
% loadat: Variable containing the mission data in the form
%         of three columns: Time(sec) RPM Torque(N-m)
% runstr: Variable containing a description of the mission.
%
clc; pause(.01);
disp('*** What is the name of this mission?')
runstr = input('? : ', 's');
disp(' ')
disp(['*** ', runstr, ' is now the current mission.'])
disp(' ')
disp('*** Would you like to save this mission')
s = input('to the "missions" directory? (y/n): ', 's');

if s == 'y' | s == 'Y'

    clc; pause(.01);
    disp(' ')
    disp('*** The following Mission Files are currently in')
    disp('the "Missions" directory')
    disp(' ')
    cd Missions
    dir *.run
    disp(' ')
    disp('*** Chose a new Mission File or one of the above to')
    disp('overwrite. Use no extension, .run will be appended.')
    disp('Return to cancel request.')
    xnewrun = input(' Example - run_no5 : ', 's');

```

```

if isempty(xnewrun) == 0
    eval(['save ',xnewrun,'.run loaddat runstr -mat']);
    disp(' ')
    disp(['*** ',xnewrun,'.run has been saved under'])
    disp('the "Missions" directory. ');
else
    disp(' ')
    disp(['*** ',runstr,' has not been saved,'])
    disp('but is still the current mission.')
end
cd ..
else
    disp(' ')
    disp(['*** ',runstr,' has not been saved,'])
    disp('but is still the current mission.')
end
end

```

E. ASCRUN.M LOADS AND SAVES AN ASCII MISSION FILE

```

function [loaddat,runstr]=ascrun(runstr);
%ASC RUN [loaddat,runstr] = ascrun(runstr)
% Load mission data from selected ASCII file in the main
%directory. This function lists the available*.???' files
%and tasks the user to chose one. The chosen file is
%loaded and the data can be displayed using SHOWRUN.
%The ASCII file can then be saved to a normal Matlab file
%in the "Missions" directory.
%
%loaddat: Variable containing the mission data in the form
% of three columns: Time(sec) RPM Torque(N-m)
%runstr: Variable containing a description of the mission.
%

clc;pause(.01);
try_again = 1;
while try_again ==1
    disp(' ')
    disp('*** The following ASCII files are available')
    disp(' ')
    dir *.???.
    disp(' ')
    disp('*** Chose an ASCII mission to convert with extension')
    disp('Or Return to cancel request.')
    xrun = input(' ? : ','s');

    if (exist(xrun) == 2)
        eval(['load ',xrun]);
        try_again = 0;
        clc;pause(.01);
        disp([' ']);
        disp(['*** ',xrun,' Loaded'])
        loaddat = eval(xrun(1:length(xrun)-4));
        [runstr] = saverun(loaddat);
    end
end

```

```

elseif isempty(xrun) == 1
    try_again = 0;
    clc; pause(.01);
    disp(' ')
    disp('*** Request Canceled')
    disp('Current Mission Data is retained')
    disp(['Mission Name: ', runstr])

else
    disp(' ')
    disp(['*** NO SUCH FILE AS: ', xrun, '!'])
end
end
end

```

F. SHOWDAT2.M DISPLAYS MISSION, BATTERY AND MOTOR DATA

```

function showdat2(loadat, runstr, modata, mostr, effmatx, battdat, batstr, batcurvs, shwhat);
%SHOWDAT2 showdat2(loadat, runstr, modata, mostr, effmatx,
%               battdat, batstr, batcurvs, shwhat)
% Displays the current data based on which "shwhat"
%is used.
%
%shwhat = 0 Shows All Data
%shwhat = 1 Shows Mission Data.
%shwhat = 2 Shows Motor Data.
%shwhat = 3 Shows Battery Data.
%shwhat = 5 Shows Motor and Battery Data along with the
%      plots of the Mission Results.
%
%loadat: Variable containing the mission data in the form
%      of three columns: Time(sec) RPM Torque(N-m)
%runstr: Variable containing a description of the mission.
%modata: Vector containing motor parameters.
%mostr:      Contains the name of the motor.
%effmatx: Matrix containing the efficiency matrix of the
%      motor if available. The torque and RPM indexes
%      are included as the first row and column.
%battdat: Vector containing battery parameters.
%batstr: The name of the battery.
%batcurvs: The discharge curves of the given battery. The
%      Percent Capacity Used and Current indexes are
%      included as the first row and column.
%

if shwhat == 1 | shwhat == 0;
    xrun = ['Current Mission Loaded: ' runstr];
    clc; pause(.001);
    disp(' ');
    disp('*** Mission Data');
    disp(' ');
    disp(xrun);
    disp(' ')

    if shwhat == 1
        disp(' STEP 3: Using the EPS "Load Battery Data")
    end
end

```

```

    disp('button, load a selected battery.')
end

```

```

curfig=gcf;
fig_1 = figure(curfig+1);
set(fig_1,'Name','Load RPM')
set(fig_1,'Position',[120 410 560 420])
plot(loadat(:,1),loadat(:,2))
grid
title(['RPM vs Time for ',runstr]);
xlabel('Time (sec)');
ylabel('RPM')

```

```

fig_2 = figure(curfig+2);
set(fig_2,'Name','Load Torque')
set(fig_2,'Position',[150 380 560 420])
plot(loadat(:,1),loadat(:,3))
grid
title(['Torque vs Time for ',runstr]);
xlabel('Time (sec)');
ylabel('Torque (N-m)')
end

```

```

if shwhat == 2 | shwhat == 0 | shwhat == 5;
    if shwhat == 2
        clc;pause(.01);
    end
    mostr1 = ['Current Motor Loaded: ' mostr];

```

```

disp('*** Motor Data')
disp([' ']);
disp(mostr1);
disp(['Number of poles:      ',num2str(modata(1,4))]);
disp(['Damping coefficient:   ',num2str(modata(1,5))]);
disp(['Inertia:                ',num2str(modata(1,6)),' K-gm^2']);
disp(['Armature resistance:    ',num2str(modata(1,7)),' ohms']);
disp(['Electrical time-constant: ',num2str(modata(1,8))]);
disp(['Back-EMF constant:      ',num2str(modata(1,9)),' V/KRPM']);
disp('Back-EMF constant:')
disp([' Peak Phase Voltage:    ',num2str(modata(1,9)),' Vpp/KRPM']);
disp([' D.C. Voltage:          ',num2str(modata(1,9)*pi/2),' Vdc/KRPM']);
disp(['Torque constant:         ',num2str(modata(1,10)),' N-m/A']);

```

```

if shwhat == 2
    disp(' ')
    disp(' The Motor Data can be altered and saved')
    disp('using the EPS "Change Motor Data" button.')
    disp(' ')
    disp('STEP 5: Using the EPS "Open Simulation Window"')
    disp('button, open the SIMULINK window that contains')
    disp('the EPS Model.')
end

```

```

if modata(3) == -1;

```



```

Tqind = effmatx(1,2:size(effmatx,2));
rpmind = effmatx(2:size(effmatx,1),1);
effmat = effmatx(2:size(effmatx,1),2:size(effmatx,2));

curfig = gcf;
fig_1 = figure(curfig+1);
set(fig_1,'Name','Motor Efficiency')
set(fig_1,'Position',[180 350 560 420])
colormap(jet)
axis([0 max(Tqind) 0 max(rpmind) 0 100])
surf(Tqind,rpmind,effmat),grid
shading flat
colorbar
xlabel('Torque'),
ylabel('RPM'),
zlabel('Eff(%)'),
title(['Motor Efficiency for ',mostr]);
end
end

if shwhat == 3 | shwhat == 0 | shwhat == 5;
if shwhat == 3
    clc;pause(.01);
end

rcapind = batcurvs(1,2:size(batcurvs,2));
lindx = batcurvs(2:size(batcurvs,1),1);
dischrg = batcurvs(2:size(batcurvs,1),2:size(batcurvs,2));

lindleg = [num2str(lindx(1),2),' Amps'];
for n = 2:length(lindx);
    lindleg = str2mat(lindleg,[num2str(lindx(n),2),' Amps']);
end

batstr1 = ['Current Battery Loaded:      ' batstr];
disp(' ');
disp('*** Battery Data')
disp(' ');
disp(batstr1);
disp(['A-h rating:                        ',num2str(battdat(4))]);
disp(['Initial State of Charge (percent) is: ',num2str(battdat(3))]);
disp(['Number of Cells in Series is:      ',num2str(battdat(1))]);

if shwhat == 3
    disp(' ')
    disp('STEP 4: Using the EPS "Load Motor Data" button')
    disp('select and load a motor.')
end

if batcurvs(1,1) ~= 998;
    curfig=gcf;
    fig_1 = figure(curfig+1);
    set(fig_1,'Name','Battery Discharge Curves')
    set(fig_1,'Position',[210 320 560 420])
    plot(rcapind,dischrg)

```

```

title(['Voltage vs Percent Capacity Used at various I for ',batstr])
xlabel('Percent Capacity Used')
ylabel('Battery Voltage')
grid
legend(lindleg)
else
    curfig=gcf;
    fig_1 = figure(curfig+1);
    set(fig_1,'Name','Battery Discharge Curves')
    set(fig_1,'Position',[210 320 560 420])
    semilogx(rcapind,dischrg)
    title(['Voltage vs Percent Capacity Used at various I for ',batstr])
    xlabel('Logarithmic Percent Capacity Used')
    ylabel('Battery Voltage')
    grid
    legend(lindleg)
end
end

```

G. WINSIZE.M ALLOWS USER TO SET MATLAB COMMAND WINDOW SIZE

```

function winsize();
%WINSIZE winsize

```

% Displays a rectangle in the Matlab Command Window
 %so that the User can set the Command window size. This
 %will ensure that the User sees all available information
 %rather than having it scroll by.

```

clc;pause(.001)
disp('*****TOP*****TOP*****TOP*****TOP*****TOP*****TOP*****TOP*****')
disp('*Left                                                                    Right*')
disp('*** Welcome to the Electric Propulsion Simulator (EPS).                *')
disp('* Default data has been loaded to initialize                          *')
disp('* all variables                                                         *')
disp('*                                                                       *')
disp('* Please make your selections from the Control Buttons                 *')
disp('* in the EPS window and follow the directions given                   *')
disp('* in the Matlab Command window.                                         *')
disp('*                                                                       *')
disp('* Prompts will be given for the next logical step,                     *')
disp('* however, selections can be made in any order since                   *')
disp('* default values have been loaded.                                       *')
disp('*                                                                       *')
disp('* STEP 1: Use the "Set Matlab Command Window Size"                     *')
disp('* button to repeat this message and display the size                   *')
disp('* of the Matlab Command Window necessary in                             *')
disp('* order to see all of the information provided.                         *')
disp('*                                                                       *')
disp('* At any time use the EPS "Show All Data"                               *')
disp('* button to display all currently loaded data                           *')
disp('*                                                                       *')
disp('*Left Side                                                                Right Side                            *')
disp('*                                                                       *')
disp('* STEP 2: Using the EPS "Load Mission" button, load a                  *')

```

```

disp('* mission to be simulated.                *')
disp('*                                          *')
disp('*Left Side                                Right Side*')
disp('*****BOTTOM*****BOTTOM*****BOTTOM*****BOTTOM*****')

```

H. GETBATT1.M RETRIEVES BATTERY DATA FROM THE "BATTERY" DIRECTORY

```

function [batcurvs,battdat,batstr]=getbatt1(battdat);
%GETBATT1 [batcurvs,battdat,batstr]=getbatt1(battdat,batstr)
% Allows the user to select a battery type which is
%loaded from the "Batterys" directory and input the
%following battery parameters:
%
% Battery Capacity (Amp-hrs) : battdat(1)
% Percent State of Charge : battdat(2)
% Number of cells in series : battdat(4)
% battdat(3) is set to 1.
%
%batcurvs:The discharge curves of the given battery. The
% Percent Capacity Used and Current indexes are
% included as the first row and column.
%battdat: Vector containing battery parameters.
%batstr: The name of the battery.

```

```

cd Batterys
s = 'n';
while (s~='y') & (s~='Y')
    clc;pause(.001)
    corinpt = 1;
    while corinpt == 1;
        disp(' ')
        disp('*****')
        disp('*** The following lines allow the user to input')
        disp('*** Battery Parameters -')
        disp('*****')
        disp(' ');
        disp('*** Choose Battery Type:')
        disp(' ')
        disp(' 1. Silver-Zinc (Ag-Zn)')
        disp(' 2. Nickel-Cadmium (Ni-Cd)')
        disp(' 3. Lithium-Sulfur (Li-SO2)')
        bat_choice = input('*** Select a Number: ');

        if bat_choice == 1
            load Ag_Zn.bat -mat;
            corinpt = 0;

        elseif bat_choice == 2
            load Ni_Cad.bat -mat;
            corinpt = 0;

        elseif bat_choice == 3
            load Li_SO.bat -mat;
            corinpt = 0;
    end
end

```

```

else
    disp(' ')
    disp(['*** ',num2str(bat_choice),' is not a valid choice!'])

end
end

clc;pause(.001)
disp(' ')
disp(['*** Battery Type: ',batstr]);
disp([' ']);
disp(['*** The current A-h rating of battery is: ',num2str(battdat(4))]);
disp(' ')
disp('Enter new A-hr rating')
inpt = input('or return to retain value: ');
if isempty(inpt)==0
    battdat(4) = inpt;
end

clc;pause(.001)
disp([' ']);
disp(['*** The current Initial State of Charge (percent) is: ',num2str(battdat(3))]);
disp(' ')
disp('Enter new Initial State of Charge (percent)')
inpt = input('or return to retain value: ');
if isempty(inpt)==0
    battdat(3) = inpt;
end

battdat(2)=1;

clc;pause(.001)
disp([' ']);
disp(['*** The current number of Cells in Series is: ',num2str(battdat(1))]);
disp(' ')
disp('Enter a new number of Cells in Series')
inpt = input('or return to retain value: ');
if isempty(inpt)==0
    battdat(1) = inpt;
end

clc;pause(.001)

disp(' ')
disp('*** Battery Parameters -')
disp(' ')
disp([' Battery Type: ',batstr]);
disp([' A-h rating: ',num2str(battdat(4))]);
disp([' Initial State of Charge (percent) is: ',num2str(battdat(3))]);
disp([' Number of Cells in Series is: ',num2str(battdat(1))]);
s = input('*** Are these Battery Parameters correct? (y/n): ','s');
end

rcapind = batcurvs(1,2:size(batcurvs,2));
batcurvs(2:size(batcurvs,1),1) = batcurvs(2:size(batcurvs,1),1)*battdat(4);

```

```
batcurvs(2:size(batcurvs,1),2:size(batcurvs,2)) =
batcurvs(2:size(batcurvs,1),2:size(batcurvs,2))*battdat(1);
```

```
clc;pause(.001)
disp(' ')
disp('*** Battery Model load complete!')
disp(' ')
disp('Battery Discharge Curves can be seen using the')
disp('EPS "Show Battery Data" button.')
disp(' ')
disp('STEP 4: Using the EPS "Load Motor Data" button')
disp('select and load a motor.')
cd ..
```

I. MAKEBAT.M ALLOWS USER TO "BUILD" A BATTERY

```
function [batcurvs,batstr]=makebat();
%MAKEBAT makebat()
% This function allows the user to create battery
% curves and save them to the "Batterys" directory.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Battery Curves Matrix Menu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;pause(.01);

disp(' ')
disp('*****')
disp('*** The following lines allow the user to input')
disp('*** the Battery Curves Matrix -')
disp('*****')
contin = input(' Continue? (y/n): ','s');
if contin == 'y' | contin == 'Y'
    disp(' ')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      User to input Battery Curves Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

s = 'n';
while (s ~= 'y') & (s ~= 'Y')
    disp('***Input Battery Curves Matrix.')
    disp(' ')
    disp('Rows of the matrix correspond to points')
    disp('of increasing Current, Columns corespond to')
    disp('point of increasing Percent Battery')
    disp('Capacity Used (PBCU). Matrix entries are')
    disp('Single Cell Voltages.')
```

```

disp(' ')
disp('Begin matrix with open bracket ([).')
disp('End matrix with close bracket (]).')
disp('Seperate elements of each row with a space.')
disp('Seperate each row with a hard return.')
disp(' ')
disp('Example:')
disp('[1 2 3')
disp('4 5 6')
disp('7 8 9]')
voltemp = input('?:');
clc;pause(.01);
disp(' ')
voltemp
s = input(' Is this Matrix correct? (y/n): ','s');
if s == 'n' | s == 'N'
    disp(' ')
    disp('***Would you like to')
    disp(' 1 = Re-enter entire matrix,')
    disp(' 2 = Make corrections by row and column')
    howch = input('?: ');
    if howch == 2
        while (s~='y') & (s~='Y')
            clc;pause(.01);
            disp(' ')
            voltemp
            numrow = input('Input row number of correction: ');
            numcol = input('Input column number of correction: ');
            newval = input('Input new value: ');
            voltemp(numrow,numcol) = newval;
            voltemp
            s = input('*** Is this Matrix correct? (y/n): ','s');
        end
    end
end
end
end

[row,col]=size(voltemp);

%%%%%%%%%%
%%%%%%%%%%
%      User to input Current Index
%%%%%%%%%%
%%%%%%%%%%
clc;pause(.01);
s = 'n';
while (s~='y') & (s~='Y')
    disp(' ')
    disp('*** Enter your Current index vector corresponding')
    disp('to the rows of your Battery Curves Matrix.')
    disp(' ')
    disp('Start the vector with an open bracket ([).')
    disp('Seperate elements with a space.')
    disp('End the vector with a close bracket (]).')
    disp(' ')
    disp(['Your vector must contain ',num2str(row),' elements'])

```

```

curtemp = input('?');
clc;pause(.01);
disp(' ')
curtemp
if length(curtemp) ~= row
    disp(' ')
    disp('*****')
    disp(' The number of vector elements does not match')
    disp(' the number of rows in the Battery Matrix!')
    disp('*****')

    s = 'n';
else
    s = input('*** Is this vector correct? (y/n): ','s');

end
end

%%%%%%%%%%%%%%
% User to input PBCU Index
%%%%%%%%%%%%%%
clc;pause(.01);
s = 'n';
while (s~='y') & (s~='Y')

    disp(' ')
    disp('*** Enter your PBCU index vector corresponding')
    disp('to the columns of your Battery Curves Matrix.')
    disp(' ')
    disp('Start the vector with an open bracket ([].)')
    disp('Seperate elements with a space.')
    disp('End the vector with a close bracket (]).)')
    disp(' ')
    disp(['Your vector must contain ',num2str(col),' elements'])
    pbtemp = input('?');
    clc;pause(.01);
    disp(' ')
    pbtemp
    if length(pbtemp) ~= col
        disp(' ')
        disp('*****')
        disp(' The number of vector elements does not match')
        disp(' the number of columns in the Battery Matrix!')
        disp('*****')
        s = 'n';
    else
        s = input('***Is this vector correct? (y/n): ','s');
    end
end

clc;pause(.01);
disp(' ')

```

```

disp('*** Should these Battery Curves be displayed on a')
disp('Normal or Semi-log plot?')
disp(' ')
disp(' 1. Normal')
disp(' 2. Semi-log')
whplot = input(' ?:');

if whplot == 1
    dumvar = 999;
else
    dumvar = 998;
end

batcurvs = [dumvar pbtemp;[curtemp' voltemp]];

clc;pause(.01);
disp(' ')
disp('*** What is the name of this Battery?')
batstr = input('*** ? : ','s');
s = input('*** Would you like to save this Battery? (y/n): ','s');

if s == 'y' | s == 'Y'

    clc;pause(.01);
    disp(' ')
    disp('*** The following Battery Files are currently in')
    disp('the "Battery" directory')
    disp(' ')
    cd Batterys
    dir *.bat
    disp(' ')
    disp('*** Chose a new Battery File or one of the above')
    disp('to overwrite.')
    disp('Use no extension, .bat will be appended.')
    disp('Return to cancel request.')
    xbat = input(' Example - Ag_Zn : ','s');

    if isempty(xbat) == 0
        eval(['save ',xbat,'.mot batstr batcurvs -mat']);
        disp(' ')
        disp(['*** ',xbat,'.bat has been saved under'])
        disp('the "Batterys" directory and is the')
        disp('currently loaded Battery.')
    else
        clc;pause(.01);
        disp(' ')
        disp(['*** ',batstr,' has not been saved,'])
        disp('but is still the currently loaded Battery.')
    end
    cd ..
else
    clc;pause(.01);
    disp(' ')
    disp(['*** ',batstr,' has not been saved,'])

```



```

    disp('but is still the currently loaded Battery.')
end
else
    disp(' ')
    disp(['*** ',batstr,' has not been saved,'])
    disp('but is still the currently loaded Battery.')
end

```

J. GETMO.M RETRIEVES MOTOR DATA FROM THE "MOTORS" DIRECTORY

```

function [modata,mostr,effmatx]=getmo(modata,mostr);
%GETMO      [modata,mostr,effmatx] = getmo(modata,mostr)
% Load motor data from selected '.mot' file in 'Motors'
%directory. This function lists the available '*.mot'
%files in the 'Motors' directory and tasks the user to
%chose one. The chosen file is loaded and the Motor
%Parameters are displayed. The Efficiency Matrix can
%be displayed using the SHOW MOTOR DATA button.
%
%modata: Vector containing motor parameters.
%mostr:      Contains the name of the motor.
%effmatx: Matrix containing the efficiency matrix of the
%          motor if available. The torque and RPM indexes
%          are included as the first row and column.

clc;pause(.001)
cd Motors
try_again = 1;
while try_again == 1
    disp(' ')
    disp('*** The following Motor Files are available')
    disp(' ')
    dir *.mot;
    disp(' ');
    disp('*** Chose a Motor File, use no extension; (.mot assumed)')
    xmotor = input(' Or Return to cancel request: ','s');
    xmotor1 = [xmotor,'.mot'];

    if (exist(xmotor1) == 2)
        eval(['load ',xmotor1,' -mat']);
        try_again = 0;
        clc;pause(.001)
        disp([' ']);
        disp(['*** ',xmotor1,' Loaded; Motor Name: ',mostr])

    elseif isempty(xmotor) == 1
        try_again = 0;
        clc;pause(.001)
        disp(' ')
        disp('*** Request Canceled')
        disp(' Current Motor Parameters are retained')

    else
        disp(' ')

```

```

disp('*****')
disp(['*** NO SUCH FILE AS: ',xmotor1, '!'])
disp('*****')
end
end

cd ..
disp(' ')
disp(['*** Motor Characteristics of: ',mostr])
disp(' ');
disp(['Number of poles: ',num2str(modata(1,4))]);
disp(['Damping coefficient: ',num2str(modata(1,5))]);
disp(['Inertia (Kgm^2): ',num2str(modata(1,6))]);
disp(['Armature resistance (ohms): ',num2str(modata(1,7))]);
disp(['Electrical time-constant: ',num2str(modata(1,8))]);
disp('Back-EMF constant:');
disp([' Peak Phase (Vpp/KRPM): ',num2str(modata(1,9))]);
disp([' D.C. (Vdc/kRPM): ',num2str(modata(1,9)*pi/2)]);
disp(['Torque constant (Nm/A): ',num2str(modata(1,10))]);
disp(' ')
disp(' The Motor Efficiency Curve can be seen using')
disp('the EPS "Show Motor Data" button.')
disp(' ')
disp(' The Motor Data can be altered and saved')
disp('using the EPS "Change Motor Data" button.')
disp(' ')
disp('STEP 5: Using the EPS "Open Simulation Window"')
disp('button, open the SIMULINK window that contains')
disp('the EPS Model.')

```

K. SETMO2.M ALLOWS USER TO CHANGE MOTOR DATA

```

function [modata,mostr,effmatx]=setmo2(modata,effmatx);
%SETMO2 [modata,mostr,effmatx]=setmo2(modata,effmatx)
% Allows user to create a Motor Data file and load the
%results. This function displays the current motor
%parameters and prompts the user for the required data
%to be entered to include:
%
% Number of poles.
% Damping coefficient.
% Inertia (Kgm^2).
% Armature resistance (ohms).
% Electrical time-constant.
% Back-EMF constant (V/KRPM).
% Torque constant (Nm/A).
%
% Entering no change retains the current motor parameter.
%
% An Efficiency Matrix based on Motor RPM and Load Torque
%is also required with the RPM index corresponding to the
%rows of the Matrix and the Torque index corresponding to
%the columns. These indexes will be included as the first
%row and column of the matrix (effmatx) with index(1,1) as
%a dummy variable = 999. If the Efficiency Matrix is

```

```

%unknown, the user may select from the following defaults:
%
% User chosen constant value Efficiency for all ranges.
% * This option must be used with caution since the
% * model will operate at this efficiency which may be
% * impossible for a real motor, thus producing
% * erroneous results.
% The Efficiency Matrix for the Unique Mobility 127.
% The Efficiency Matrix for the Unique Mobility 86.
% Copper Loss only calculated by the motor model. This
% selection results in no efficiency matrix
% generated.
% Retain the current Efficiency Matrix
%
% To use these default Efficiency Matrices a RPM and
%Torque range must be given to calculate the appropriate
%indexes.
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User Enters Motor Characteristics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;pause(.01);
disp([' ']);
disp('*****')
disp('*** The following lines allow the user to input')
disp('*** New Motor Characteristics -')
disp('*** ')
disp('*** Enter new data or return to retain current values.')
disp('*****')
disp([' ']);

disp(['*** The current Number of Poles (P) is: ',num2str(modata(1,4))]);
inpt = input(' Enter new Number of Poles: ');
if isempty(inpt)==0
    modata(1,4) = inpt;
end

clc;pause(.01);
disp([' ']);
disp(['*** The current Damping Coefficient (B) is: ',num2str(modata(1,5))]);
inpt = input(' Enter new Damping Coefficient: ');
if isempty(inpt)==0
    modata(1,5) = inpt;
end

clc;pause(.01);
disp([' ']);
disp(['*** The current Inertia (J) is: ',num2str(modata(1,6)),' Kgm^2']);
inpt = input(' Enter new Inertia (Kgm^2): ');
if isempty(inpt)==0
    modata(1,6) = inpt;
end

```

```

clc;pause(.01);
disp([' ']);
disp(['*** The current Armature Resistance (ra) is: ',num2str(modata(1,7)),' ohms']);
inpt = input(' Enter new Armature Resistance (ohms): ');
if isempty(inpt)==0
    modata(1,7) = inpt;
end

clc;pause(.01);
disp([' ']);
disp(['*** The current Electrical Time Constant (ta) is: ',num2str(modata(1,8))]);
inpt = input(' Enter new Electrical Time Constant: ');
if isempty(inpt)==0
    modata(1,8) = inpt;
end

clc;pause(.01);
disp([' ']);
disp(['*** The current Back-EMF Constant (ke) is:']);
disp([' for Peak Phase Voltage ',num2str(modata(1,9)),' Vpp/KRPM']);
disp([' or for D.C. Voltage ',num2str(modata(1,9)*pi/2),' Vdc/KRPM']);
disp(' ')
disp(['*** The current Torque Constant (kt) is: ',num2str(modata(1,10)),' N-m/A']);
disp(' ')
disp(' The Back-EMF Constant and the Torque Constant')
disp('are related by a constant so that only one of')
disp('them needs to be entered.')
disp(' ')
whatvar = 1;
while whatvar == 1;
    disp(' 1. Enter new Back-EMF Constant (V/KRPM)')
    disp(' 2. Enter new Torque Constant (N-m/A)')
    disp(' ')
    disp('Choose the number of the variable to enter,')
    newvar = input('or return to retain current values: ');
    if isempty(newvar) ~= 0;
        whatvar = 0;
    elseif newvar == 1;
        clc;pause(.01);
        whatke = 1;
        while whatke == 1;
            disp(' ')
            disp(['*** It is imperative that the correct number be'])
            disp(['entered as the Back-EMF Constant. The EPS uses the'])
            disp(['Peak Phase Voltage Back-EMF Constant, but will convert'])
            disp(['a DC Voltage Back-EMF Constant by multiplying it by 2/pi.'])
            disp(['Please answer the following question about the'])
            disp(['Constant you are about to enter.'])
            disp(' If the value of the constant was measured as')
            disp(['or generated by a D. C. Voltage, pick choice 1.'])
            disp(' If the value of the constant was measured as')
            disp(['a peak phase voltage pick choice 2.'])
            disp(' ')
            disp(' Does your Back-EMF Constant relate KRPM to')
            disp(' ')
        end
    end
end

```

```

disp(' 1. A D. C. Voltage?')
disp(' 2. A Peak Phase Voltage?')
newke = input(' Choose a number (1 or 2): ');
inpt = input(' Enter new Back-EMF Constant (V/KRPM): ');
if isempty(inpt)==0
    if newke == 1;
        modata(1,9) = inpt*(2/pi);
        whatke = 0;
    elseif newke == 2;
        modata(1,9) = inpt;
        whatke = 0;
    else
        disp('*** You must choose option 1 or 2 !!!')
    end
end
end
modata(1,10) = modata(1,9)*.001*30*1.5/pi;
whatvar = 0;
elseif newvar == 2
    inpt = input(' Enter new Torque Constant (N-m/A): ');
    if isempty(inpt)==0
        modata(1,10) = inpt;
    end
    modata(1,9) = modata(1,10)*1000*pi/45;
    whatvar = 0;
else
    disp('*** Not a valid entry!')
end
end

s = 'n';
while (s~='y') & (s~='Y')
    clc; pause(.01);
    disp(' ')
    disp('*** Motor Characteristics -')
    disp([' ']);
    disp([' 1 Number of poles: ', num2str(modata(1,4))]);
    disp([' 2 Damping coefficient: ', num2str(modata(1,5))]);
    disp([' 3 Inertia (Kgm^2): ', num2str(modata(1,6))]);
    disp([' 4 Armature resistance (ohms): ', num2str(modata(1,7))]);
    disp([' 5 Electrical time-constant: ', num2str(modata(1,8))]);
    disp(' 6 Back-EMF constant:')
    disp([' Peak Phase (Vpp/KRPM): ', num2str(modata(1,9))]);
    disp([' D.C. (Vdc/kRPM): ', num2str(modata(1,9)*pi/2)]);
    disp([' 6 Torque constant (Nm/A): ', num2str(modata(1,10))]);
    disp(' ')
    disp('*** Chose one line number to correct,');
    moch = input('or enter a zero(0) if all correct: ');
    disp(' ')

    if moch == 0
        s = 'y';
    elseif moch == 1
        clc; pause(.01);

```

```

disp(' ')
disp(['*** The current Number of poles (P) is: ',num2str(modata(1,4))]);
inpt = input(' Enter new Number of poles: ');
if isempty(inpt)==0
    modata(1,4) = inpt;
end

elseif moch == 2
    clc;pause(.01);
    disp(' ')
    disp(['*** The current Damping Coefficient (B) is: ',num2str(modata(1,5))]);
    inpt = input(' Enter new Damping Coefficient: ');
    if isempty(inpt)==0
        modata(1,5) = inpt;
    end

elseif moch == 3
    clc;pause(.01);
    disp(' ')
    disp(['*** The current Inertia (J): ',num2str(modata(1,6))]);
    inpt = input(' Enter new Inertia (Kgm^2): ');
    if isempty(inpt)==0
        modata(1,6) = inpt;
    end

elseif moch == 4
    clc;pause(.01);
    disp(' ')
    disp(['*** The current Armature Resistance (ra): ',num2str(modata(1,7))]);
    inpt = input(' Enter new Armature Resistance (ohms): ');
    if isempty(inpt)==0
        modata(1,7) = inpt;
    end

elseif moch == 5
    clc;pause(.01);
    disp(' ')
    disp(['*** The current Electrical Time Constant (ta): ',num2str(modata(1,8))]);
    inpt = input(' Enter new Electrical Time Constant: ');
    if isempty(inpt)==0
        modata(1,8) = inpt;
    end

elseif moch == 6
    clc;pause(.01);
    disp(' ');
    disp('*** The current Back-EMF Constant (ke) is:');
    disp([' for Peak Phase Voltage ',num2str(modata(1,9)), ' Vpp/KRPM']);
    disp([' or for D.C. Voltage ',num2str(modata(1,9)*pi/2), ' Vdc/KRPM']);
    disp(' ')
    disp(['*** The current Torque Constant (kt) is: ',num2str(modata(1,10)), ' N-m/A']);
    disp(' ')
    disp(' The Back-EMF Constant and the Torque Constant');
    disp('are relatedby a constant so that only one of them');
    disp('needs to be entered.')
    disp(' ')

```

```

whatvar = 1;
while whatvar == 1;
    disp(' 1. Enter new Back-EMF Constant (V/KRPM)')
    disp(' 2. Enter new Torque Constant (N-m/A)')
    disp(' ')
    disp('Choose the number of the variable to enter,')
    newvar = input('or return to retain current values: ');
    if isempty(newvar) ~= 0;
        whatvar = 0;
    elseif newvar == 1;
        clc; pause(.01);
        whatke = 1;
        while whatke == 1;
            disp(' ')
            disp('*** It is imperative that the correct number be')
            disp('entered as the Back-EMF Constant. The EPS uses the')
            disp('Peak Phase Voltage Back-EMF Constant, but will convert')
            disp('a DC Voltage Back-EMF Constant. Please answer the')
            disp('following question about the Constant you are about')
            disp('to enter.')
            disp(' If the value of the constant was measured as')
            disp('or generated by a D. C. Voltage, pick choice 1.')
            disp(' If the value of the constant was measured as')
            disp('a peak phase voltage pick choice 2.')
            disp(' ')
            disp(' Does your Back-EMF Constant relate KRPM to')
            disp(' ')
            disp(' 1. A D. C. Voltage?')
            disp(' 2. A Peak Phase Voltage?')
            newke = input(' Choose a number (1 or 2): ');
            inpt = input(' Enter new Back-EMF Constant (V/KRPM): ');
            if isempty(inpt) == 0
                if newke == 1;
                    modata(1,9) = inpt*(2/pi);
                    whatke = 0;
                elseif newke == 2;
                    modata(1,9) = inpt;
                    whatke = 0;
                else
                    disp('*** You must choose option 1 or 2 !!!')
                end
            end
        end
        end
        modata(1,10) = modata(1,9)*.001*30*1.5/pi;
        whatvar = 0;
    elseif newvar == 2
        inpt = input('Enter new Torque Constant (N-m/A): ');
        if isempty(inpt) == 0
            modata(1,10) = inpt;
        end
        modata(1,9) = modata(1,10)*1000*pi/45;
        whatvar = 0;
    end
end
end
end
end

```

```

%%%%%%%%%%
%%%%%%%%%%
%      Efficiency Matrix Menu
%%%%%%%%%%
%%%%%%%%%%

```

```

clc;pause(.01);
effchk = 1;
while effchk == 1;

```

```

    disp(' ')
    disp('*****')
    disp('*** The following lines allow the user to input')
    disp('*** the Efficiency Matrix -')
    disp('*****')
    disp(' ')
    disp('*** Efficiency Matrix: Select one of the following:')
    disp('or Return to keep current Efficiency Matrix')
    disp(' ')
    disp(' 1 - User inpputed values')
    disp(' 2 - User inpputed constant value')
    disp(' *** CAUTION ***')
    disp(' This option must be used with caution as the model')
    disp(' will operate at this efficiency at all operating points!')
    disp(' This may produce incorrect results!')
    disp(' 3 - Unique Mobility 86 Efficiency')
    disp(' 4 - Unique Mobility 127 Efficiency')
    disp(' 5 - Calculated Efficiency for Copper Loss only')
    disp(' This option does not generate an Efficiency Matrix.')
    disp(' The Copper Loss will be calculated by the motor model.')
    disp(' 6 - Keep current Efficiency Matrix')
    witeff=input(' Enter choice: ');

```

```

if isempty(witeff)==1
    witeff = 6;
end

```

```

if witeff == 4 | witeff == 3 | witeff == 2
    clc;pause(.01);
    disp(' ')
    rpmmin = input('Enter minimum RPM: ');
    rpmmax = input('Enter maximum RPM: ');
    trqmin = input('Enter minimum Torque (N-m): ');
    trqmax = input('Enter maximum Torque (N-m): ');

```

```

    rpmind = linspace(rpmmin,rpmmax,10);
    Tqind = linspace(trqmin,trqmax,10);
    effchk = 0;

```

```

if witeff == 4
    effmat=[10 20 32 39 38 40 51 41 28 18;
            18 36 60 71 70 69 72 63 55 45;
            20 50 76 85.5 82 80.5 79 75 71 64;
            30 60 85.5 86 85.5 84.8 83 82 79 74;
            33 71 86 87.5 86.5 86.5 85.5 85 83.5 81;

```



```

38 80 87 88.5 89 88 87 86 95.5 84;
40 84 89 90 90.5 90 89 88 87 86;
50 85 89.5 91 92 91.5 91 90 90.5 89.5;
50 86 90 92 93 93.5 91 91.5 90.5 89.5;
48 86.5 90.5 92.5 94 93.5 93 92 91 90.5];

```

```

elseif witeff == 3
    effmat=[35 55 67 68 69 70 71 70 63 55;
            58 68 72 74 75 74 72 70 68 66;
            68 74 78 78.5 78.6 76 74 72 70 69;
            72 77 82 82 81.9 81 80 77 73 70;
            74 79 84 84.4 85 84.8 83 82 80 78;
            78 80 86 87.3 87.2 87.1 86.5 84.5 83 81.5;
            81 84 89 89.7 90.1 89.8 88 87 86 85.5;
            82 86 90.5 92 92.2 92.8 91 89.2 88.5 87;
            83 87 91.7 94 93.5 92.5 91.6 89.8 88.6 87.1;
            84 87.5 93 94.5 94.2 93 91.9 90 89.5 88.5];

```

```

elseif witeff == 2
    clc;pause(.01);
    disp(' ')
    disp('*** THIS OPTION MUST BE USED WITH CAUTION!')
    disp('The model will operate at this efficiency even though')
    disp('it may be impossible for a real motor to do so.')
    disp(' ')
    disp('For example:')
    disp('the User could enter an efficiency > 100 Percent')
    disp('and the model would operate at that efficiency!')
    disp(' ')
    effval = input('Input constant Efficiency value (0 - 100): ');
    effmat = effval*ones(10);
end

```

```

effmatx = [999 Tqind;[rpmind' effmat]];
modata(1) = min(min(effmat));
modata(2) = max(max(effmat));
modata(3) = -1;

```

```

elseif witeff == 1

```

```

    clc;pause(.01);
    disp(' ')
    %%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%
    % User to input Entire Efficiency Matrix
    %%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%
    s = 'n';
    while (s~='y') & (s~='Y')
        disp('***Input Efficiency Matrix.')
        disp(' ')
        disp('Rows of the matrix correspond to points')
        disp('of increasing RPM, Columns corespond to')

```

```

disp('point of increasing Torque. Matrix')
disp('entries are efficiency values')
disp(' ')
disp('Begin matrix with open bracket ([).')
disp('End matrix with close bracket (]).')
disp('Seperate elements of each row with a space.')
disp('Seperate each row with a hard return.')
disp(' ')
disp('Example:')
disp('[1 2 3')
disp('4 5 6')
disp('7 8 9]')
efftemp = input('?:');
clc;pause(.01);
disp(' ')
efftemp
s = input(' Is this Matrix correct? (y/n): ','s');
if s == 'n' | s == 'N'
    disp(' ')
    disp('***Would you like to')
    disp(' 1 = Re-enter entire matrix,')
    disp(' 2 = Make corrections by row and column')
    howch = input('?: ');
    if howch == 2
        while (s~='y') & (s~='Y')
            clc;pause(.01);
            disp(' ')
            efftemp
            numrow = input('Input row number of correction: ');
            numcol = input('Input column number of correction: ');
            newval = input('Input new value: ');
            efftemp(numrow,numcol) = newval;
            efftemp
            s = input('*** Is this Matrix correct? (y/n): ','s');
        end
    end
end
end
end

[row,col]=size(efftemp);

%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%      User to input RPM Index
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
clc;pause(.01);
s = 'n';
while (s~='y') & (s~='Y')
    disp(' ')
    disp('*** Enter your RPM index vector corresponding')
    disp('to the rows of your Efficiency Matrix.')
    disp('Start the vector with an open bracket ([).')
    disp('Seperate elements with a space.')
    disp('End the vector with a close bracket (]).')
    disp(' ')

```

```

disp(['Your vector must contain ',num2str(row),' elements'])
rpmtemp = input('?');
clc;pause(.01);
disp(' ')
rpmtemp
if length(rpmtemp) ~= row
    disp(' ')
    disp('*****')
    disp(' The number of vector elements does not match')
    disp(' the number of rows in the Efficiency Matrix!')
    disp('*****')
    s = 'n';
else
    s = input('*** Is this vector correct? (y/n): ','s');
end
end

%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%      User to input Torque Index
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
clc;pause(.01);
s = 'n';
while (s~='y') & (s~='Y')
    disp(' ')
    disp('*** Enter your Torque index vector corresponding')
    disp('to the columns of your Efficiency Matrix.')
    disp('Start the vector with an open bracket ([].)')
    disp('Seperate elements with a space.')
    disp('End the vector with a close bracket (].)')
    disp(' ')
    disp(['Your vector must contain ',num2str(col),' elements'])
    tqtemp = input('?');
    clc;pause(.01);
    disp(' ')
    tqtemp
    if length(tqtemp) ~= col
        disp(' ')
        disp('*****')
        disp(' The number of vector elements does not match')
        disp(' the number of columns in the Efficiency Matrix!')
        disp('*****')
        s = 'n';
    else
        s = input('***Is this vector correct? (y/n): ','s');
    end
end

effmatx = [999 tqtemp;rpmtemp' efftemp];
modata(1) = min(min(efftemp));
modata(2) = max(max(efftemp));
modata(3) = -1;

effchk = 0;

```

```

elseif witeff == 5
    modata(1) = 0;
    modata(2) = 0;
    modata(3) = 1;
    effchk = 0;
    effmatx = [];

elseif witeff == 6
    effchk = 0;
    effmat = effmatx(2:size(effmatx,1),2:size(effmatx,2));
    modata(1) = min(min(effmat));
    modata(2) = max(max(effmat));
    modata(3) = -1

else
    disp('*** Not a valid request!!!')
end

end    % End While Loop

clc;pause(.01);
disp(' ')

if witeff == 5
    disp('*** There will be no Efficiency Matrix associated')
    disp('with this motor. The "Copper Losses" will be')
    disp('taken into account during the simulation.')

elseif witeff == 6
    disp('*** The currently loaded Efficiency Matrix')
    disp('has been retained.')
end

disp('*** What is the name of this motor?')
mostr = input('*** ?: ','s');
s = input('*** Would you like to save this motor? (y/n): ','s');

if s == 'y' | s == 'Y'

    clc;pause(.01);
    disp(' ')
    disp('*** The following Motor Files are currently in')
    disp('the "Motors" directory')
    disp(' ')
    cd Motors
    dir *.mot
    disp(' ')
    disp('*** Chose a new Motor File or one of the above to overwrite.')
    disp('Use no extension, .mot will be appended.')
    disp('Return to cancel request.')
    xmotor = input(' Example - Uniq86 : ','s');

    if isempty(xmotor) == 0

```

```

eval(['save ',xmotor,'.mot mostr modata effmatx -mat']);
disp(' ')
disp(['*** ',xmotor,'.mot has been saved under'])
disp('the "Motors" directory and is the')
disp('currently loaded motor.')
else
    clc;pause(.01);
    disp(' ')
    disp(['*** ',mostr,' has not been saved,'])
    disp('but is still the currently loaded motor.')
end
cd ..
else
    disp(' ')
    disp(['*** ',mostr,' has not been saved,'])
    disp('but is still the currently loaded motor.')
end
end

```

L. HELPER.M PROVIDES HELP FOR EPS BUTTONS

```

function helper();
%HELPER helper()
% This function lists all of the available buttons and
%provides a brief description of each.

```

```

clc;pause(.01);
disp(' ')
disp('*** Chose an Item for Help')
disp(' or Return to cancel request.')
disp(' ')
disp(' 1. Load Mission')
disp(' 2. Save Mission')
disp(' 3. Load/Save ASCII Miss')
disp(' 4. Show Mission Data')
disp(' 5. Set Window Size')
disp(' 6. Load Battery Data')
disp(' 7. Make Battery')
disp(' 8. Show Battery Data')
disp(' 9. Show All Data')
disp(' 10. Load Motor Data')
disp(' 11. Change Motor Data')
disp(' 12. Show Motor Data')
disp(' 13. Help')
disp(' 14. Open Simulation Window')
disp(' 15. Mission Results Summary')
disp(' 16. Save Mission Results')
disp(' 17. Load Previous Results')
disp(' 18. Close')

whathelp = input(' *** Select a number: ');

if whathelp == 1;

    clc;pause(.01);

```

```

disp('*** Load Mission')
disp(' ')
disp(' This button is used to load a set of RPM and Torque Data.')
disp('A list of available missions in the missions directory will')
disp('be displayed to the User in the MATLAB command window.')
disp('The User is requested to select one of the available files')
disp('or hit Return to cancel the request. All available files')
disp('will have a ".run" extension. Once the request is made')
disp('the data will be loaded into the MATLAB workspace. The data')
disp('consists of a matrix of three columns of Time, RPM, and')
disp('Torque, and a mission name.')

elseif whathelp == 2;

    clc;pause(.01);
    disp('*** Save Mission')
    disp(' ')
    disp(' This button allows the User to save a mission into the')
    disp('"Missions" directory. This provision allows the User to')
    disp('make changes to an existing mission and then save it for')
    disp('future use. Mission data is contained in the variable')
    disp('"loadmat" consisting of three columns Time, RPM, Torque.')
    disp('The User will be asked to name the new mission and')
    disp('provide a file name. The file will then be given the')
    disp('extension ".run".')

elseif whathelp == 3;

    clc;pause(.01);
    disp('*** Load/Save ASCII Mission')
    disp(' ')
    disp(' This button allows the User to convert an ASCII file')
    disp('to MATLAB format and save it to the "Missions" directory.')
    disp('The program will look for any file in the main directory')
    disp('with a three letter extension. The user is requested to')
    disp('choose one of the files. The file is then loaded into the')
    disp('MATLAB workspace. The User is then requested to name the')
    disp('mission and then is asked if the mission is to be saved.')

elseif whathelp == 4;

    clc;pause(.01);
    disp('*** Show Mission Data')
    disp(' ')
    disp(' This button gives the RPM and Torque vs Time plots')
    disp('for the current mission loaded in the MATLAB workspace.')

elseif whathelp == 5;

    clc;pause(.01);
    disp('*** Set Window Size')
    disp(' ')
    disp(' This button displays an outline in the Matlab Command')
    disp('Window so that the Window can be adjusted to the proper')
    disp('size. This sizing is necessary to ensure that all')
    disp('available information is displayed to the User and')

```

```

disp('not hidden because of automatic scrolling.')

elseif whathelp == 6;

    clc:pause(.01);
    disp('*** Load Battery Data')
    disp(' ')
    disp(' This button allows the User to select the Battery')
    disp('Parameters prior to the simulation start. Default')
    disp('parameters have been loaded with the start of the')
    disp('Simulator program. The parameters include:')
    disp(' ')
    disp(' Battery Type, ')
    disp(' Battery Capacity, ')
    disp(' State of Charge, ')
    disp(' Cells in Series, ')

elseif whathelp == 7;
    clc:pause(.01);
    disp('*** Make Battery')
    disp(' ')
    disp(' This button allows the user to "build" a battery')
    disp('and save it to the "Batterys" directory')
    disp(' ')
    disp(' A Battery Matrix based on Current drawn and Percent')
    disp('Battery Capacity Used (PBCU). The Current index')
    disp('corresponding to the rows the Matrix and')
    disp('the PBCU index corresponding to the columns.')
    disp('These indexes will be included as the first row and')
    disp('column of the matrix (batcurvs) with index(1,1) as a dummy')
    disp('variable = 999 for normal plots or 998 for semi-log plots.')

elseif whathelp == 8;

    clc:pause(.01);
    disp('*** Show Battery Data')
    disp(' ')
    disp(' This button displays the Battery Parameters currently')
    disp('as well as generates a plot of the discharge characteristics')
    disp('for a set of current values. The current values are')
    disp('displayed in the Command Window.')

elseif whathelp == 9;

    clc:pause(.01);
    disp('*** Show All Data')
    disp(' ')
    disp(' This button displays all of the currently loaded data.')
    disp('This data includes, Mission, Battery and Motor Data. Plots')
    disp('are generated to display the Mission RPM and Torque vs time,')
    disp('Battery discharge curves and the Motor Efficiency if ')
    disp('available.')

elseif whathelp == 10;

```

```

clc;pause(.01);
disp('*** Load Motor Data')
disp(' ')
disp(' This button allows the User to select a motor file from')
disp('the "Motors" directory. The motor data is then displayed')
disp('to include:')
disp(' ')
disp(' Number of poles      ')
disp(' Damping coefficient   ')
disp(' Inertia (Kgm^2)        ')
disp(' Armature resistance (ohms)')
disp(' Electrical time-constant ')
disp(' Back-EMF constant (V/KRPM) ')
disp(' Torque constant (Nm/A) ')
disp(' ')
disp(' An Efficiency Matrix is also loaded with the motor file')
disp('and can be viewed using the "Show Motor Data" button. ')

elseif whathelp == 11;

clc;pause(.01);
disp('*** Change Motor Data')
disp(' ')
disp(' Allows user to create a Motor Data file and load the')
disp('results. This function displays the current motor parameters')
disp('and prompts the user for the required data to be entered to')
disp('include:')
disp(' ')
disp(' Number of poles.')
disp(' Damping coefficient. ')
disp(' Inertia (Kgm^2). ')
disp(' Armature resistance (ohms).')
disp(' Electrical time-constant.')
disp(' Back-EMF constant (V/KRPM).')
disp(' Torque constant (Nm/A).')
disp(' ')
disp(' Entering no change retains the current motor parameter.')
disp(' ')
disp(' An Efficiency Matrix based on Motor RPM and Load Torque')
disp('is also required with the RPM index corresponding to the')
disp('rows the Matrix and the Torque index corresponding to the')
disp('columns. These indexes will be included as the first row and')
disp('column of the matrix (effmatx) with index(1,1) as a dummy')
disp('variable = 999. If the Efficiency Matrix is unknown, the')
disp('user may select from several defaults.')

elseif whathelp == 12;

clc;pause(.01);
disp('*** Show Motor Data')
disp(' ')
disp(' This button displays the Motor Parameters currently')
disp('loaded as well as generates the Motor's Efficiency Plot')
disp('if available.')

```



```

elseif whathelp == 13;

    clc:pause(.01);
    disp('*** Help')
    disp(' ')
    disp(' Allows the User to access a Help Menu for each of the ')
    disp('EPS buttons.')

elseif whathelp == 14;

    clc:pause(.01);
    disp('*** Open Simulation Window')
    disp(' ')
    disp(' This button opens up a Simulink Window that contains the')
    disp('EPS Model. Directions concerning the operation of the model')
    disp('will be displayed in the Command Window.')

elseif whathelp == 15;

    clc:pause(.01);
    disp('*** Mission Results Summary')
    disp(' ')
    disp(' This button provides the User with a summary of the')
    disp('current mission once the simulation has completed or was ')
    disp('stopped. The summary information is generated from')
    disp('the output of the simulation, but can only be accessed')
    disp('after the simulation is complete.')

elseif whathelp == 16;

    clc:pause(.01);
    disp('*** Save Mission Results')
    disp(' ')
    disp(' This button allows the User to save all information to a')
    disp('result file in the "Results" directory. Once the data is')
    disp('saved, it can be retrieved and displayed using the EPS "Load")
    disp('Previous Results" button.')

elseif whathelp == 17;

    clc:pause(.01);
    disp('*** Load Previous Results')
    disp(' ')
    disp(' This button allows the User to load saved results from a')
    disp('previous mission and to display all the plots associated')
    disp('with that mission. This feature prevents the User from')
    disp('having to run redundant missions for the same results.')

elseif whathelp == 18;

    clc:pause(.01);
    disp('*** Close')
    disp(' ')
    disp(' This button closes the EPS Window.')

```

```

else
    clc;pause(.01);
    disp(' ')
    disp('*** ERROR')
    disp(' ')
    disp([' ',num2str(whatshelp),' => That number has no help available.'])
end

```

M. OPENMOD.M OPENS THE EPS SIMULINK WINDOW

```

function openmod();
%OPENMOD openmod()
% This function opens the Simulink window containing
%the Electric Propulsion Model and provides operating
%instructions in the Matlab Command Window.

inwork7;
clc;pause(.01);
disp(' ')
disp('*** Directions for Model simulation')
disp(' ')
disp('1. Begin simulation by selecting the "Simulation"')
disp('menu in the SIMULINK Window with the Left Mouse Button.')
disp(' ')
disp('2. Next, select the "Start" option from the menu.')
disp(' ')
disp('3. Once the Plot Windows have been generated,')
disp('it is best to select the "Pause" option from')
disp('the "Simulation" menu, then re-arrange the Plot')
disp('Windows as desired. This action prevents the')
disp('simulation from crashing, which has been known to')
disp('happen on MS Windows machines.')
disp(' ')
disp('4. Once the Plot Windows have been arranged')
disp('as desired, select the "Continue" option from')
disp('the "Simulation" menu.')
disp(' ')
disp('*****')
disp(' Once the simulation is complete, a Mission Summary')
disp('can be generated using the EPS "Mission Results Summary"')
disp('button.')

```

N. MISSSUM.M PROVIDES THE MISSION SUMMARY AFTER THE SIMULATION

```

function misssum(battdat,resdata);
%MISSSUM misssum(battdat,resdata)
% This function provides a summary of important data
%after the simulation is complete, or once a previously
%run mission is loaded. The values are calculated using
%all of the generated results of the simulation which are
%contained in the matrix "resdata". Entries is "resdata"
%correspond to the following columns:
%

```

```

% 1 Simulation Time
% 2 True Motor Model Efficiency 3 Efficiency Matrix pts
% 4 Load RPM                    5 Motor RPM
% 6 Load Torque                  7 Motor Torque
% 8 Motor Current                9 Battery Efficiency
% 10 Battery Capacity Used       11 Time when Vmot > Vbat
% 12 Delta Voltage when Vmot > Vbat
% 13 Battery Capacity Borrowed while Vmot > Vbat
% 14 Battery Voltage            15 Motor Voltage
%
% battdat: Vector of battery parameters.
%     battdat(1): Battery Capacity (Amp-Hrs)
%     battdat(2): Initial State of Charge
%     battdat(4): Number of Cells in Series
%     battdat(3): Not Used, Set to 1

clktime = resdata(:,1);
motcur = resdata(:,8);
bmvolts = resdata(:,14:15);
battcap = resdata(:,10);
capbar = resdata(:,13);
batteff = resdata(:,9);
timedelv = resdata(:,11);

avecur = sum(motcur)/length(motcur);
avevol = sum(bmvolts(:,2))/length(bmvolts);
aveeff = sum(resdata(:,3))/length(resdata(:,3));
[maxcur,indmcurr] = max(motcur);
[maxvolts,indmvolts] = max(bmvolts);
[minvolts,indmnvlt] = min(bmvolts);
maxAH = max(capbar);
finvolts = bmvolts(length(bmvolts),1);
finbattcap = battcap(length(battcap));
bateffavg = sum(batteff)/length(batteff);
maxdelv = max(resdata(:,12));
tottime = clktime(length(clktime));
delttime = timedelv(length(timedelv));
pertime = 100*delttime/tottime;
zeroind = find(timedelv == 0);
sizedelv = length(resdata(:,12))-length(zeroind);
avgdelv = sum(resdata(:,12))/sizedelv;

clc
disp('*** Mission Result Summary')
disp(' ');
disp([' Max Motor Current: ',num2str(maxcur),' Amps at Time: ',num2str(clktime(indmcurr)),
seconds'])
disp([' Average Motor Current: ',num2str(avecur),' Amps'])
disp([' Max Motor Voltage: ',num2str(maxvolts(2)), ' Volts at Time: ',
num2str(clktime(indmvolts(2))), ' seconds'])
disp([' Average Motor Voltage: ',num2str(avevol),' Volts'])
disp([' Average Motor Efficiency: ',num2str(aveeff),' Percent'])
disp([' Final Battery Voltage: ',num2str(finvolts),' Volts'])
disp([' Percent Battery Capacity Used: ',num2str(finbattcap),' Percent'])
disp([' Average Battery Efficiency: ',num2str(bateffavg),' Percent'])

```

```

if maxvolts(2) > maxvolts(1)
    numcells = (maxvolts(2)-maxvolts(1))/(maxvolts(1)/battdat(1));
    disp(' ')
    disp('*** Maximum Motor Voltage EXCEEDS Maximum Battery Voltage!')
    disp([' ',num2str(numcells),' Additional Battery Cells REQUIRED!!!!'])
end

```

```

if maxAH > 0 & finbattcap >= 80
    disp(' ')
    disp('*** More than 80 percent of battery capacity')
    disp('*** was used!!!')
    disp('AND')
    disp('*** Motor Voltage exceeded Battery voltage!!!')
    disp([' by a Maximum of: ',num2str(maxdelv),' Volts']);
    disp([' for an Average of: ',num2str(avgdelv),' Volts']);
    disp([' for: ',num2str(pertime),' Percent of the time']);
    disp([' or for: ',num2str(delttime),' Seconds']);
    disp([' Approximately, ',num2str(maxAH),' additional Amp Hours'])
    disp(' are required!')
end

```

```

if maxAH > 0 & finbattcap <= 20
    [maxdelv,indmv] = max(resdata(:,12));
    numcell = maxdelv/(bmvolts(indmv,1)/battdat(1));
    disp(' ')
    disp('*** Less than 20 percent of battery capacity')
    disp('*** was used!!!')
    disp('BUT')
    disp('*** Motor Voltage exceeded Battery voltage!!!')
    disp([' by a Maximum of: ',num2str(maxdelv),' Volts']);
    disp([' for an Average of: ',num2str(avgdelv),' Volts']);
    disp([' for: ',num2str(pertime),' Percent of the time']);
    disp([' or for: ',num2str(delttime),' Seconds']);
    disp([' Approximately, ',num2str(numcell),' additional Cells'])
    disp(' are required!')
end

```

```

if maxAH > 0 & finbattcap > 20 & finbattcap < 80
    maxdelv = max(resdata(:,12));
    disp(' ')
    disp('*** Between 20 and 80 percent of battery capacity')
    disp('*** was used!!!')
    disp('AND')
    disp('*** Motor Voltage exceeded Battery voltage!!!')
    disp([' by a Maximum of: ',num2str(maxdelv),' Volts']);
    disp([' for an Average of: ',num2str(avgdelv),' Volts']);
    disp([' for: ',num2str(pertime),' Percent of the time']);
    disp([' or for: ',num2str(delttime),' Seconds']);
    disp(' ')
    disp('Some combination of additional Battery Capacity and/or')
    disp('Cells is required.')
end

```

```

disp(' ')
disp('LAST STEP: Use the EPS "Save Mission Results" button')

```

```
disp('to save all the mission data into the "Results" directory.')
```

O. MISSSAVE.M SAVES MISSION RESULTS IN THE "RESULTS" DIRECTORY

```
function misssave(loaddat,runstr,batstr,battdat,batcurvs,mostr,modata,effmatx,resdata);
%MISSSAVE misssave(loaddat,runstr,batstr,battdat,batcurvs,mostr,
%      modata,effmatx,resdata)
% Saves mission information and results of the mission in 'Results'
%directory. This allows the user to reload the results of a previous
%mission instead of having to re-run the simulation.
%
%loaddat: Variable containing the mission data in the form
%      of three columns: Time(sec) RPM Torque(N-m)
%runstr: Variable containing a description of the mission.
%battdat: Vector containing battery parameters.
%batstr: The name of the battery.
%batcurvs:The discharge curves of the given battery. The
%      Percent Capacity Used and Current indexes are
%      included as the first row and column.
%modata: Vector containing motor parameters.
%mostr:      Contains the name of the motor.
%effmatx: Matrix containing the efficiency matrix of the
%      motor if available. The torque and RPM indexes
%      are included as the first row and column.
%resdata: Entries in "resdata" correspond to the following
%columns:
%
% 1 Simulation Time
% 2 True Motor Model Efficiency 3 Efficiency Matrix values
% 4 Load RPM                      5 Motor RPM
% 6 Load Torque                    7 Motor Torque
% 8 Motor Current                   9 Battery Efficiency
%10 Battery Capacity Used          11 Time when Vmot > Vbat
%12 Delta Voltage when Vmot > Vbat
%13 Battery Capacity Borrowed while Vmot > Vbat
%14 Battery Voltage                15 Motor Voltage

clc;pause(.01)
s = input('*** Would you like to save these mission results? (y/n): ','s');

if s == 'y' | s == 'Y'

    clc;pause(.001)
    disp(' ')
    disp('*** Enter a Mission Results Description.')
    resname = input(' : ','s');
    disp(' ')
    disp('*** The following Mission Results Files are currently in')
    disp('the "Results" directory')
    disp(' ')
    cd Results
    dir *.res
    disp(' ')
    disp('*** Chose a mission result file name without an extension')
    disp('or Return to cancel request.')
```

```

newresults = input(' Example - res_no5 : ','s');

if isempty(newresults) == 0
    eval(['save ',newresults,'.res resname loaddat runstr batstr battdat batcurvs mostr modata
effmatx resdata -mat']);
    cd ..
    disp(' ')
    disp(['*** ',newresults,'.res'])
    disp(['Description: ',resname])
    disp('Has been saved under the "Results" directory. ');
end

else
    disp('*** Mission Results have not been saved!')
end

disp(' ')
disp('*** Would you like to save the current mission, motor')
disp('and battery combination as the default load. This')
disp('option will allow these parameters to be loaded')
disp('automatically the next time the EPS is run.')
newdeflt = input('(y/n): ','s');

if newdeflt == 'y' | newdeflt == 'Y'
    clc;pause(.01);
    cd Missions
    save deflt22 loaddat runstr batcurvs battdat batstr modata mostr effmatx
    cd ..
    disp(' ')
    disp('*** New Startup Default (deflt22.mat) has been saved to the')
    disp('"'Missions" directory')
else
    clc;pause(.01);
    disp(' ')
    disp('*** New default not saved')
end

```

P. MISSLOAD.M LOADS PREVIOUS RESULTS FROM "RESULTS" DIRECTORY

```

function [loaddat,runstr,batstr,battdat,batcurvs,mostr,modata,effmatx,resdata] = missload();
%MISSLOAD [loaddat,runstr,batstr,battdat,batcurvs,mostr,
%      modata,effmatx,resdata]=missload()
% This function loads previously run mission results
%instead of having to re-run the simulation.
%
%loaddat: Variable containing the mission data in the form
%      of three columns: Time(sec) RPM Torque(N-m)
%runstr: Variable containing a description of the mission.
%battdat: Vector containing battery parameters.
%batstr: The name of the battery.
%batcurvs:The discharge curves of the given battery. The
%      Percent Capacity Used and Current indexes are
%      included as the first row and column.
%modata: Vector containing motor parameters.

```

```

%mostr:          Contains the name of the motor.
%effmatx: Matrix containing the efficiency matrix of the
%      motor if available. The torque and RPM indexes
%      are included as the first row and column.
%resdata: Entries is "resdata" correspond to the following
%columns:
%
% 1 Simulation Time
% 2 True Motor Model Efficiency 3 Efficiency Matrix values
% 4 Load RPM                    5 Motor RPM
% 6 Load Torque                  7 Motor Torque
% 8 Motor Current                9 Battery Efficiency
% 10 Battery Capacity Used       11 Time when Vmot > Vbat
% 12 Delta Voltage when Vmot > Vbat
% 13 Battery Capacity Borrowed while Vmot > Vbat
% 14 Battery Voltage            15 Motor Voltage

cd Results
clc;pause(.001)
try_again = 1;
while try_again == 1
    disp(' ')
    disp('*** The following Previous Mission Results are available')
    disp(' ')
    dir *.res;
    disp(' ')
    disp('*** Chose a result to load, use no extension; (.res assumed)')
    xres1 = input(' Or Return to cancel request: ','s');
    xres = [xres1, '.res'];

    if (exist(xres) == 2)
        eval(['load ',xres,' -mat']);
        clc;pause(.001)
        disp(' ')
        disp([' File Loaded: ',xres]);
        disp([' Mission Name: ',runstr])
        disp([' Description: ',resname])
        try_again = 0;

        dispdat = input('*** Would you like results displayed (10 plots)? (y/n): ','s');

        if (dispdat == 'y') | (dispdat == 'Y')

            clc;pause(.01)
            disp(' ')
            disp([' File Loaded: ',xres]);
            disp([' Mission Name: ',runstr])
            disp([' Description: ',resname])

            showdat2(loaddat,runstr,modata,mostr,effmatx,battdat,batstr,batcurvs,5);

            curfig = gcf;

            fig_1 = figure(curfig+1);
            set(fig_1,'Name','Load and Motor RPM')
            set(fig_1,'Position',[240 290 560 420])

```

```

plot(resdata(:,1),resdata(:,4:5))
legend('Load RPM','Motor RPM')
title(['Load and Motor RPM vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Load and Motor RPM')
grid

fig_2 = figure(curfig+2);
set(fig_2,'Name','Load and Motor Torque')
set(fig_2,'Position',[270 260 560 420])
plot(resdata(:,1),resdata(:,6:7))
legend('Load Torque','Motor Torque')
title(['Load and Motor Torque vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Load and Motor Torque (N-m)')
grid

fig_3 = figure(curfig+3);
set(fig_3,'Name','Motor Current Required for Mission')
set(fig_3,'Position',[300 230 560 420])
plot(resdata(:,1),resdata(:,8))
title(['Motor Current vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Motor Current (Amps)')
grid

fig_4 = figure(curfig+4);
set(fig_4,'Name','Battery and Motor Voltages')
set(fig_4,'Position',[330 200 560 420])
plot(resdata(:,1),resdata(:,14:15))
legend('Battery Voltage','Motor Voltage')
title(['Battery and Motor Voltage vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Battery and Motor Voltage (Volts)')
grid

fig_5 = figure(curfig+5);
set(fig_5,'Name','True and Matrix Efficiencies')
set(fig_5,'Position',[360 170 560 420])
plot(resdata(:,1),resdata(:,2:3))
legend('True','Matrix')
title(['True and Matrix Efficiency vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Motor Efficiency')
grid

fig_6 = figure(curfig+6);
set(fig_6,'Name','Percent Battery Capacity Used')
set(fig_6,'Position',[390 140 560 420])
plot(resdata(:,1),resdata(:,10))
title(['Percent Battery Capacity Used vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Percent Capacity Used')
grid

```



```

fig_7 = figure(curfig+7);
set(fig_7,'Name','Excessive Amp-Hours Barrowed')
set(fig_7,'Position',[420 110 560 420])
plot(resdata(:,1),resdata(:,13))
title(['Excessive Amp-Hours ( $V_{mot} > V_{bat}$ ) vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Amp-Hours')
grid

fig_8 = figure(curfig+8);
set(fig_8,'Name','Battery Efficiency')
set(fig_8,'Position',[450 80 560 420])
plot(resdata(:,1),resdata(:,9))
title(['Battery Efficiency ( $V_{bat}/V_{oc}$ ) vs Time for ',runstr])
xlabel('Time in seconds')
ylabel('Battery Efficiency')
grid

end

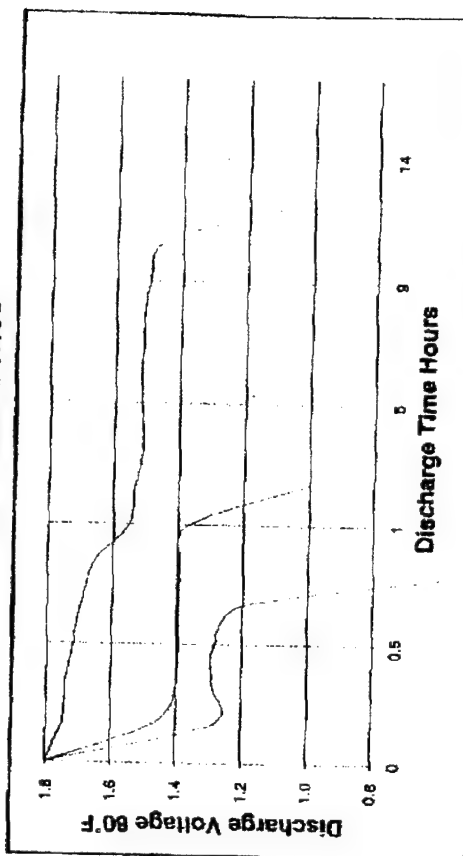
elseif isempty(xres) == 0
    try_again = 0;
    clc; pause(.001)
    disp(' ')
    disp('*** Request Canceled')

else
    disp(' ')
    disp('*****')
    disp(['*** NO SUCH FILE AS: ',xres,'.res!'])
    disp('*****')
end
end
cd ..

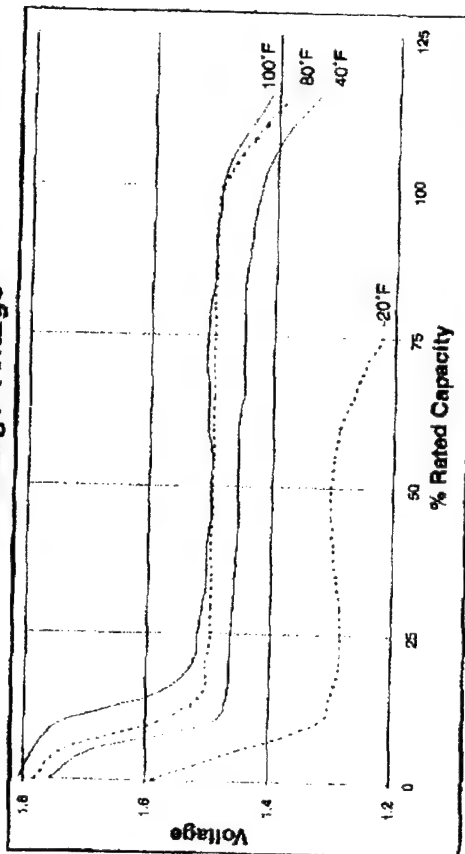
```

APPENDIX C. MANUFACTURER'S DATA SHEETS

Typical Capacity and Voltage Characteristics Low Rate Series

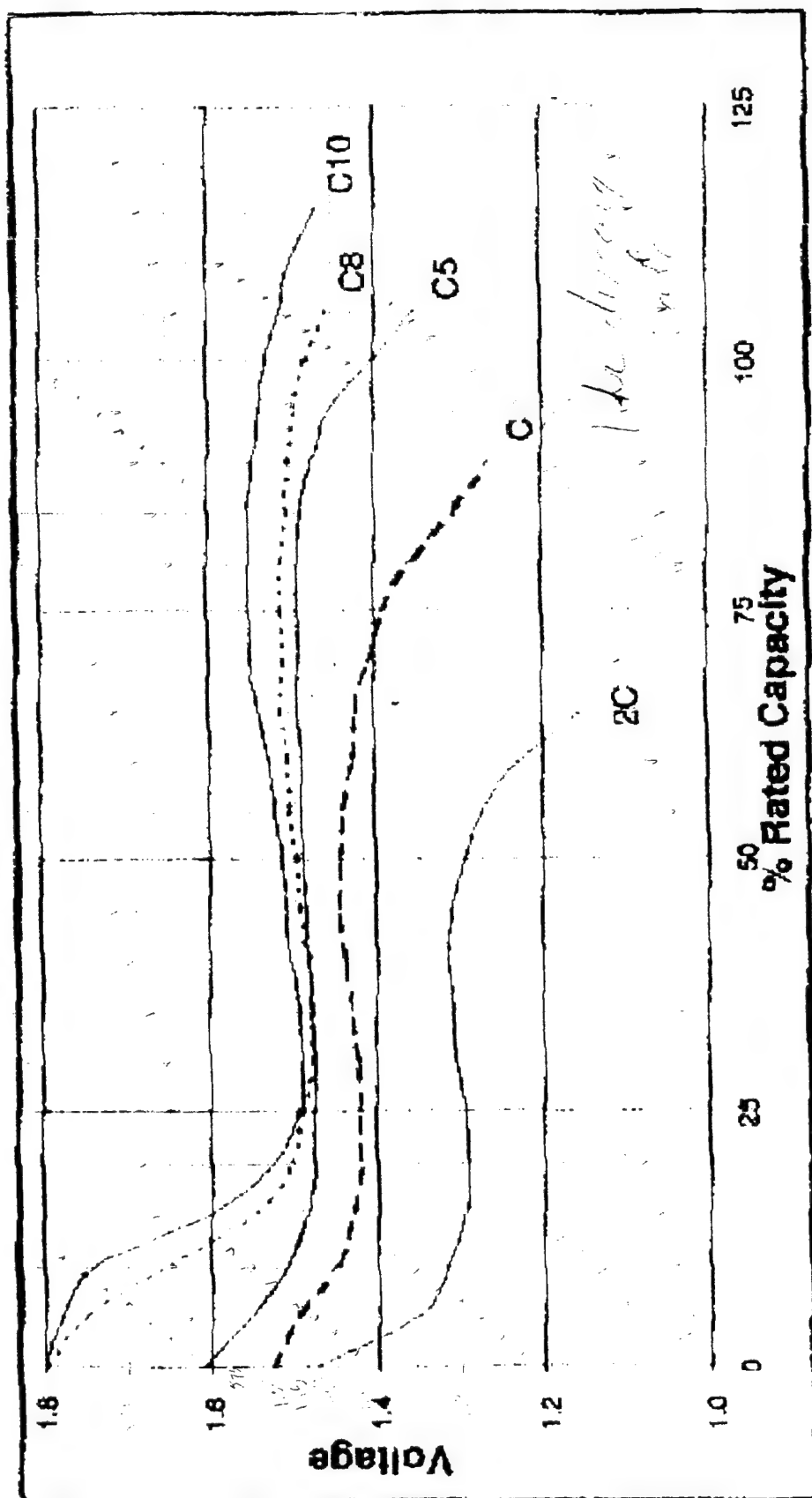


Effect of Temperature on Discharge Voltage



Silver-Zinc Systems

Typical Cell Discharge Characteristics at Various Rates (80°F)





MANUFACTURING AND SALES

890 W. 23rd ST. ■ HIALEAH, FL 33010
(305) 884-4040 ■ FAX (305) 884-3483

February 9, 1996

Naval Post Graduate School
Curric. Office Rm 404 Code 32
833 Dyer Road
Monterey, CA 93943-5120

Attn: Joel Yourkowski, ECE Program

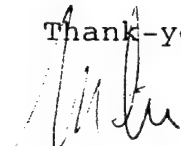
Joel:

Please Let us take this opportunity to introduce ourselves to you. We are PRO BATTERY SPECIALISTS, serving the international community with competitive pricing and extremely high levels of customer service and satisfaction.

PRO BATTERY SPECIALISTS is in a very good position to be a dependable long-range source and supply for all your battery and specialty lamp requirements. As always, a phone call will get you our current competitive prices

If you have any questions or if I may be of any further service to you, please feel free to contact me at your convenience.

Thank-you,


Mike Krasner

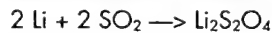
Saft 3.0 V system: LO series

Initially developed to provide reliable continuous high currents for communications use with the US Military, these cells are now in use in more militaries throughout the world than any other. This is due to the excellent safety, reliability, low cost, high flat voltage curve, temperature insensitivity and ten-year shelf life demonstrated in the Saft design. Because of the leverage of the US Military adoption of lithium sulphur dioxide, it is now becoming the best solution for low cost or high power lithium battery needs for industrial and commercial applications.

Chemistry

Anode: lithium (Li) - Cathode: sulphur dioxide (SO₂)
Electrolyte: sulphur dioxide with lithium bromide and acetonitrile

Electrochemical reaction

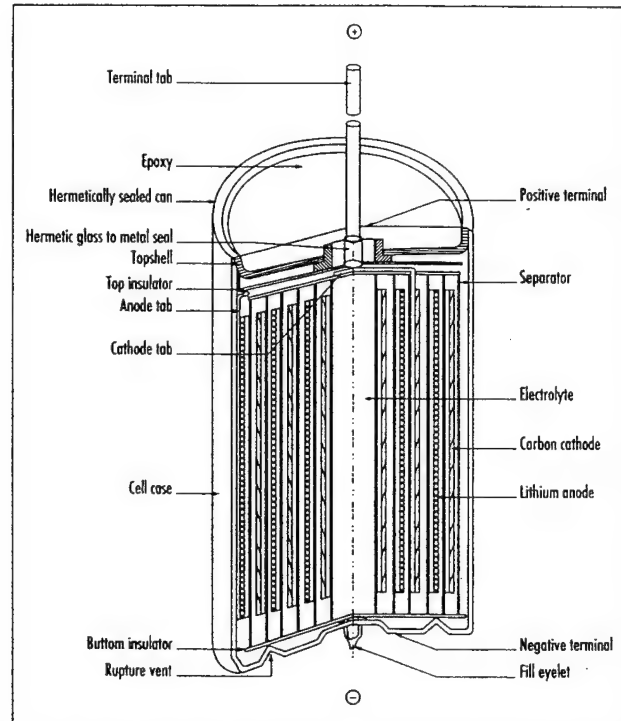


Design: Cell electrode design: spiral wound

Container material: nickel plated steel

Sealing system: TA-23 glass to metal feedthrough

Safety: rupture vent

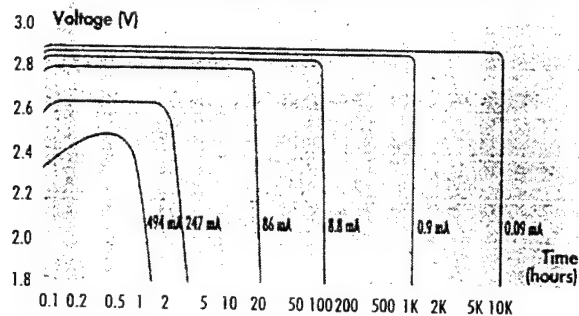


		LO14-SX	LO33-SX	LO95-SX	LO151-SX	LO151-SX	LO151-SX	LO151-SX	LO151-SX	LO151-SX	LO151-SX
Size	IEC	1/3R14	2/3R14	R14	—	—	R20	R20	R20	—	—
	ANSI	1/3C	2/3C	C	2/3 thin D	thin D	D	D	D	fat D	F
Nominal capacity at 20 °C/68 °F/2 V cut-off	Ah	1.0	2.0	3.5	3.5	5.7	8.0	6.5	7.5	8.5	12.5
		c/30	c/30	c/30	c/30	c/30	c/30	c/30	c/30	c/30	c/30
Open circuit voltage (at 20 °C)	volts	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
Nominal voltage at 20 °C/68 °F/2 V cut-off	volts	2.8	2.8	2.8	2.8	2.8	2.8	2.8	2.7	2.8	2.8
		c/30	c/20	c/30	c/20	c/2	c/30	c/2	c/2	c/30	c/13
Maximum recommended constant current	A	0.5	1.2	1.5	2.0	2.0	4.0	pulses up to 30 A	4.0	8.0	8.0
Operating	°C	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71
	°F	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160
Storage	°C	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71	-60/+71
	°F	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160	-76/+160
Diameter (max.)	mm	25.6	25.6	25.6	29.1	29.1	33.8	33.8	33.8	38.7	31.4
	in	1.010	1.010	1.010	1.145	1.145	1.33	1.33	1.33	1.525	1.236
Height (max.)	mm	19.1	35.6	50.3	36.0	59.9	59.0	59.0	59.0	50.0	99.5
	in	0.750	1.40	1.980	1.420	2.360	2.324	2.324	2.324	1.97	3.92
Weight	g	16	30	40	40	63	85	85	85	96	125
	oz	0.56	1.05	1.40	1.40	2.24	2.98	2.98	2.98	3.40	4.40
UL Recognition		•	•	•	•	•	•	•	•	•	•

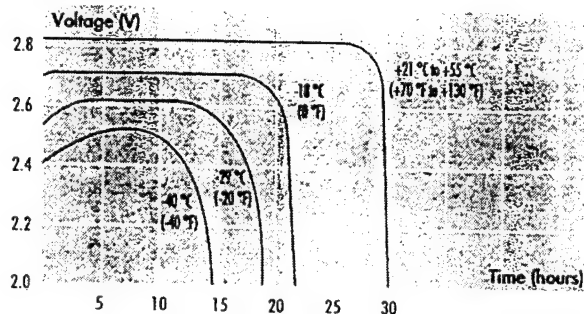
LO SX and SHX are balanced cells (safe in reversal) - LO SH and SHX: cells optimized for high rate performance (1.5 V cut-off)

Typical discharge curves of LO batteries

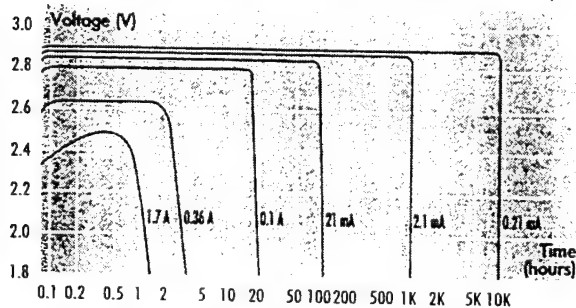
LO34 SX - Operating voltage vs drain (21 °C/70 °F)



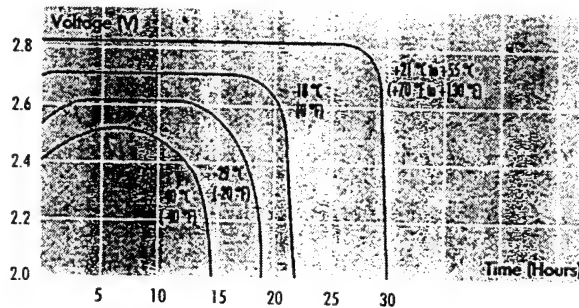
LO34 SX - Discharge profiles
(I = 33 mA)



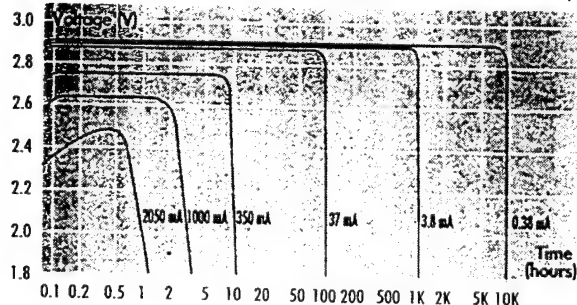
LO35 SX - Operating voltage vs drain (21 °C/70 °F)



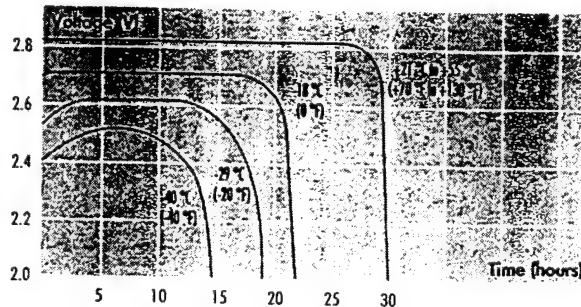
LO35 SX - Discharge profiles
(I = 66 mA)



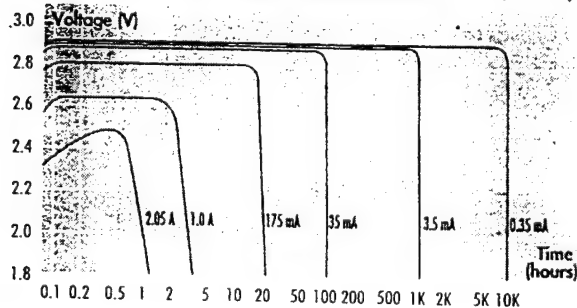
LO29 SHX - Operating voltage vs drain (21 °C/70 °F)



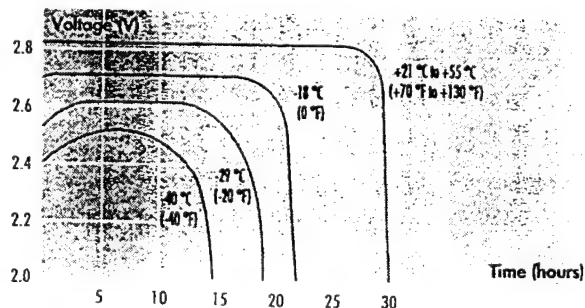
LO29 SHX - Discharge profiles
(I = 117 mA)



LO40 SHX - Operating voltage vs drain (21 °C/70 °F)

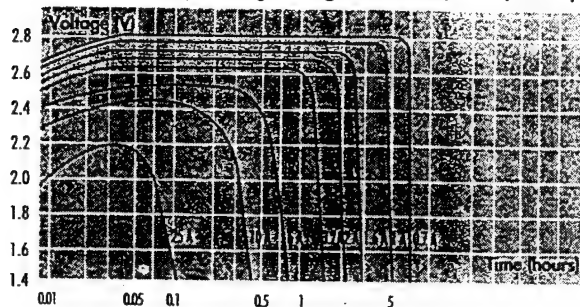


LO40 SHX - Discharge profiles
(I = 117 mA)

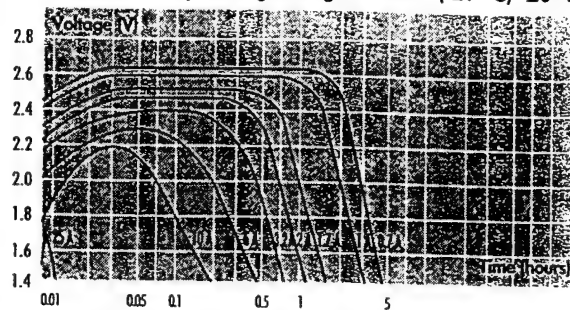


Typical discharge curves of LO batteries

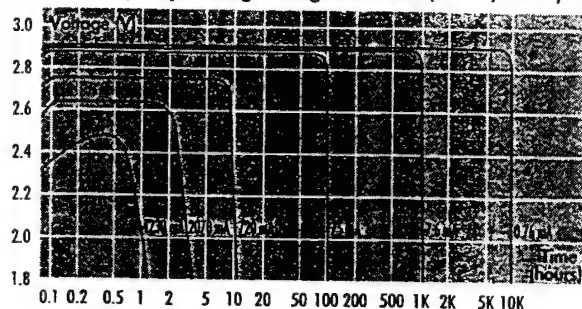
LO30 SHX - Operating voltage vs drain (21 °C/70 °F)



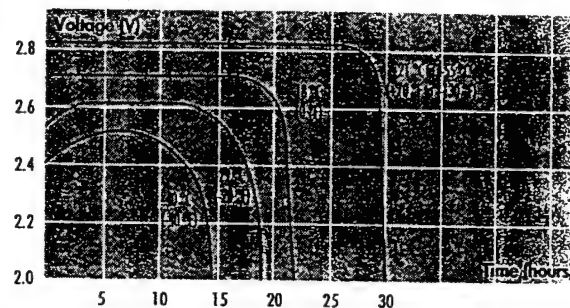
LO30 SHX - Operating voltage vs drain (-29 °C/-20 °F)



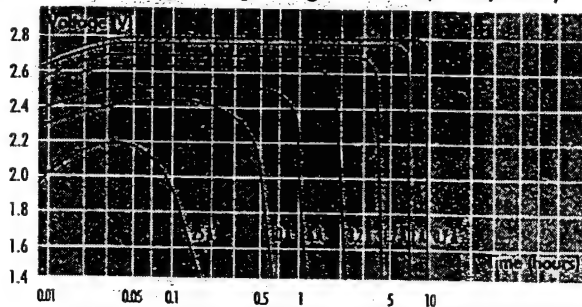
LO26 SX - Operating voltage vs drain (21 °C/70 °F)



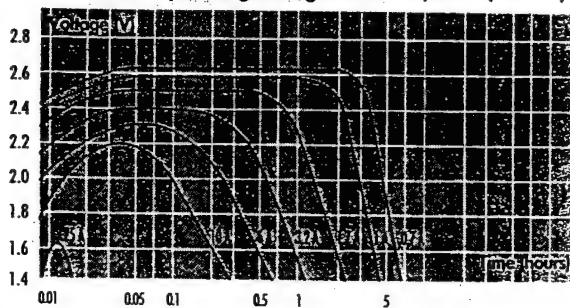
LO26 SX - Discharge profiles
(I = 0.24 A)



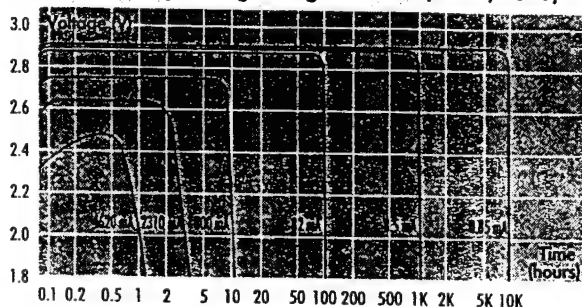
LO26 SH - Operating voltage vs drain (21 °C/70 °F)



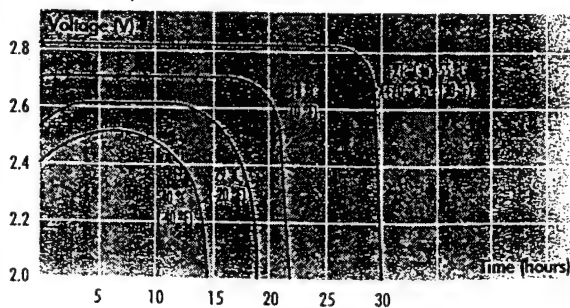
LO26 SH - Operating voltage vs drain (-29 °C/-20 °F)



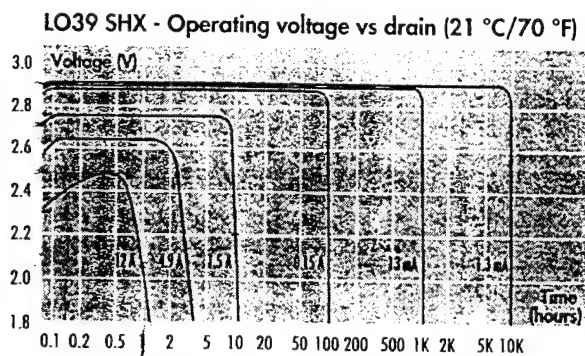
LO25 SX - Operating voltage vs drain (21 °C/70 °F)



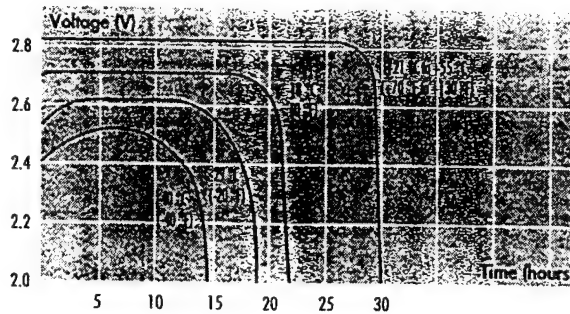
LO25 SX - Discharge profiles
(I = 0.267 A)



Typical discharge curves of LO batteries



LO39 SHX - Discharge profiles
(I = 0.4 A)



SR BATTERIES, INC.**FAX MEMO**

BOX 287, BELLPORT, NEW YORK 11713 • PHONE 516-286-0079 • FAX 516-286-0901

FROM: LARRY SRIBNICK
ONE PAGETO: Joel Yourkowski
DATE: February 9, 1996

Dear Joel,

I think this is the information you're looking for for our 1500 Max Series cell. Let me know if you have any questions.

Cell Volts, 1.5 Amp Load	
Time in Minutes	Voltage
10	1.26
20	1.25
30	1.24
40	1.22
50	1.19
60	1.13
63	1.00

Cell Volts, 3 Amp Load	
Time in Minutes	Voltage
10	1.245
20	1.22
30	1.1
31	1.00

Cell Volts, 6 Amp Load	
Time in Minutes	Voltage
2	1.23
4	1.22
6	1.15
8	1.20
10	1.18
12	1.15
14	1.09
14.5	1.0

Cell Volts, 12 Amp Load	
Time in Minutes	Voltage
1	1.20
2	1.19
3	1.17
4	1.16
5	1.15
6	1.10
6.5	1.00

% Cell Capacity	
Load in Amps	%
.10	100
1.5	97
3	94
4.5	92
6	89
7.5	86
9	84

LIST OF REFERENCES

1. Roerig, S. J., "Simulation of a Solar Powered Electric Vehicle Under the Constraints of the World Solar Challenge," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.
2. Krause, J. C., Wasynczuk, O., Sudhoff, S. D., *Analysis of Electric Machinery*, McGraw Hill, New York 1986.
3. "Simulink Users Guide," The MathWorks Inc., Natick, Massachusetts, 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
 8725 John J. Kingman Rd., STE 0944
 Ft. Belvoir VA 22060-6218

2. Dudley Knox Library 2
 Naval Postgraduate School
 411 Dyer Rd.
 Monterey CA 93943-5101

3. Director, Training and Education 1
 MCCDC, Code C46
 1019 Elliot Rd.
 Quantico VA 22134-5027

4. Chairman, Code EC 1
 Department of Electrical and Computer Engineering
 Naval Postgraduate School
 Monterey CA 93943-5121

5. Professor Jovan E. Lebaric, Code EC/Jb 1
 Department of Electrical and Computer Engineering
 Naval Postgraduate School
 Monterey CA 93943-5121

6. Professor John S. Ciezki, Code EC/Cy 1
 Department of Electrical and Computer Engineering
 Naval Postgraduate School
 Monterey CA 93943-5121

7. Maj Joel Yourkowski 1
 P. O. Box 727
 Quantico VA 22134-9998